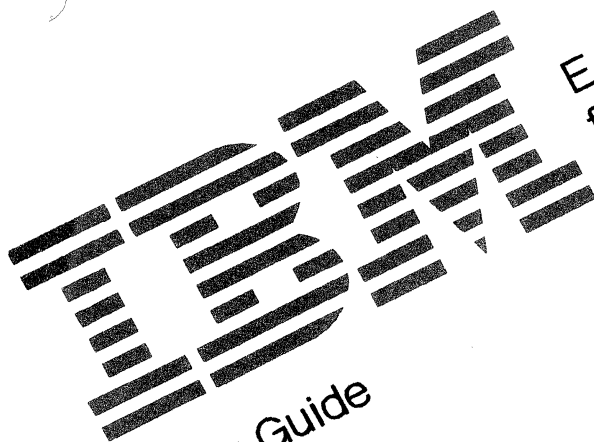




Developer's Guide
Volume 2

EASEL Version 1.1
for OS/2
Extended Edition

IBM PROGRAM LICENSE AGREEMENT

BEFORE OPENING THIS PACKAGE, YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS. OPENING THIS PACKAGE INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS. IF YOU DO NOT AGREE WITH THEM YOU SHOULD PROMPTLY RETURN THE PACKAGE UNOPENED AND YOUR MONEY WILL BE REFUNDED.

This is a license agreement and not an agreement for sale. IBM owns, or has licensed from the owner, copyrights in the Program. You obtain no rights other than the license granted you by this Agreement. Title to the enclosed copy of the Program, and any copy made from it, is retained by IBM. IBM licenses your use of the Program in the United States and Puerto Rico. You assume all responsibility for the selection of the Program to achieve your intended results and for the installation of, use of, and results obtained from, the Program.

The Section in the enclosed documentation entitled "License Information" contains additional information concerning the Program and any related Program Services.

LICENSE

You may:

1. use the Program on only one machine at any one time, unless permission to use it on more than one machine at any one time is granted in the License Information (Authorized Use);
2. make a copy of the Program for backup or modification purposes only in support of your Authorized Use. However, Programs marked "Copy Protected" limit copying;
3. modify the Program and/or merge it into another program only in support of your Authorized Use; and
4. transfer possession of copies of the Program to another party by transferring this copy of the IBM Program License Agreement, the License Information, and all other documentation along with at least one complete, unaltered copy of the Program. You must, at the same

time, either transfer to such other party or destroy all your other copies of the Program, including modified copies or portions of the Program merged into other programs. Such transfer of possession terminates your license from IBM. Such other party shall be licensed, under the terms of this Agreement, upon acceptance of this Agreement by its initial use of the Program.

You shall reproduce and include the copyright notice(s) on all such copies of the Program, in whole or in part.

You shall not:

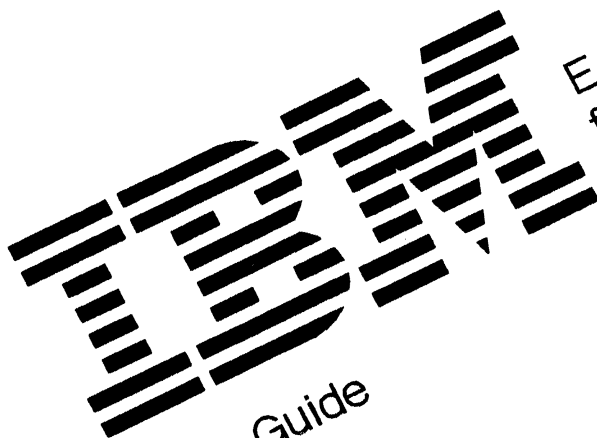
1. use, copy, modify, merge, or transfer copies of the Program except as provided in this Agreement;
2. reverse assemble or reverse compile the Program; and/or
3. sublicense, rent, lease, or assign the Program or any copy thereof.

LIMITED WARRANTY

Warranty details and limitations are described in the Statement of Limited Warranty which is available upon request from IBM, its Authorized Dealer or its approved supplier and is also contained in the License Information. IBM provides a three-month limited warranty on the media for all Programs. For selected Programs, as indicated on the outside of the package, a limited warranty on the Program is available. The applicable Warranty Period is measured from the date of delivery to the original user as evidenced by a receipt.

Certain Programs, as indicated on the outside of the package, are not warranted and are provided "AS IS."

Continued on inside back cover.



Developer's Guide
Volume 2

EASEL Version 1.1
for OS/2
Extended Edition

Second Edition (May 1990)

This publication could include technical inaccuracies or typographical errors. Changes are made periodically to the information herein; these changes will be incorporated in new editions of this publication.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

Products and publications are not stocked at the address given below. Requests for copies of this product and for technical information about the system should be made to your local IBM representative.

A form for reader's comments is provided at the back of this publication. If the form has been removed, address your comments to IBM Corporation, Software Development, Dept. 877, 10401 Fernwood Road, Bethesda, Maryland 20817. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1990. All rights reserved.

© Copyright Interactive Images, Inc. 1989, 1990.

Note to US Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

IBM and OS/2 are registered trademarks of International Business Machines Corporation.

EASEL is a registered trademark of Interactive Images, Inc.

Preface

IBM EASEL® for OS/2® Extended Edition, which is based on IBM Operating System/2™ (OS/2) Presentation Manager™, is a graphical user interface development facility. It is comprised of two licensed programs for developing and executing graphical user interface applications on IBM OS/2 Extended Edition programmable workstations. IBM EASEL for OS/2 Extended Edition (referred to as EASEL OS/2 EE) can support stand-alone applications, the workstation portion of cooperative processing applications, and applications that are front-ends to existing host terminal-based applications.

This publication is meant to help graphical user interface application developers, whose goal is to overcome end-user training and retention time, learn to write EASEL OS/2 EE graphical user interface applications. Although the publication does not assume previous computer graphics experience, you should be familiar with basic programming concepts (such as variables and loops) and IBM OS/2 Extended Edition. Before proceeding, you should be comfortable using simple OS/2 commands (such as TYPE and CD) and an OS/2 text editor.

EASEL OS/2 EE is developed by Interactive Images, Inc., Woburn, MA.

IBM and OS/2 are registered trademarks of International Business Machines Corporation.

EASEL is a registered trademark of Interactive Images, Inc.

Operating System/2 and Presentation Manager are trademarks of International Business Machines Corporation.

This publication is organized as follows:

Chapter	1	Installation
	2	Welcome to EASEL OS/2 EE
	3	Programming with EASEL OS/2 EE
	4	Language Elements
	5	Objects, Coordinate Systems, and Fonts
	6	Responses, Blocks, Action Statements, and Classes
	7	Graphical Objects, Image Regions, and Sense Regions
	8	Textual Regions
	9	Dialog Boxes, Dialog Regions, and Dialog Controls
	10	Region Attributes and Action Bars
	11	Program Flow and Performance
	12	Layout/CUA™
	13	Library Subroutines and Functions
	14	Application Communications, PM Clipboard, and Stimulus
	15	Controlling the Keyboard and Pointing Devices
	16	Business Graphics
	17	SQL Support
	18	Audio Support
	19	APPC Support
	20	3270 Support
	21	5250 Support
	22	Printing and Plotting
	23	Environment Files
	24	Debugging
	25	Writing Local Applications and Libraries
Appendix	A	Include Files
	B	Specified Operating Environment
	C	Fatal Runtime Errors

The accompanying *IBM EASEL Version 1.1 for OS/2 Extended Edition Language Reference Manual* provides a summary of the EASEL OS/2 EE language statements and functions, in alphabetical order. The *IBM EASEL Version 1.1 for OS/2 Extended Edition Quick Reference Card* provides a quick reference to the EASEL OS/2 EE syntax of commonly used language statements and functions.

Layout/CUA is a trademark of Interactive Images, Inc.

Notation Conventions

The syntax models of EASEL OS/2 EE language statements and functions presented in this publication use the conventions shown below.

Convention	Meaning	Examples
<code>^</code>	Indicates a control character.	<code>^D</code>
bold lowercase words	EASEL OS/2 EE statement name or keyword. You must specify exactly as shown.	region selected line is checked
bold symbols	You must specify exactly as shown.	<code>()</code> <code>[]</code> <code>:</code>
UPPERCASE WORDS	Value or identifier you specify. You must substitute your own identifiers or values.	<code>OBJECT_NAME</code> <code>LOOP_NAME</code> <code>LINE_NO</code>
Square brackets <code>[]</code>	Optional entry. You can optionally specify the entry in brackets.	<code>[primary]</code> <code>[column]</code>
	Multiple choice optional entry. You can select one or none of the entries in the brackets. Choices are separated by a vertical bar (<code> </code>).	<code>[guarded resumable]</code>
	When used to specify elements and dimensions in array definitions and references, you must specify exactly as shown.	string <code>ARRAY[N]</code>
Underlined keywords	The default. Does not need to be specified.	<code>[<u>enabled</u> disabled]</code>

Convention	Meaning	Examples
Braces { }	Required entry. You must select one of the entries in the braces. Choices are separated by a vertical bar ().	{ keyboard APPLICATION }
Ellipses ... (horizontal)	Optional repetition. You can repeat the preceding bracketed entry any number of times. When used without a preceding bracketed entry, indicates an omitted portion of syntax.	[ACTION_STATEMENT]... [in PARENT_NAME2]... response to border
: (vertical)	Omitted code. Portions of the EASEL OS/2 EE code have been omitted.	begin Block_Name : end Block_Name

(

(

Identifiers and Color Values

In all syntax models, except compile-time definitions, an *identifier* can be replaced with a string value representing the identifier. Compile-time definitions are those definitions not created during runtime with the **add** action statement.

In all syntax models, a compile-time definition *identifier* can be replaced with a compile-time string value representing the identifier.

In all syntax models, **COLOR** is a string value representing a color keyword, an integer value representing a color number, or a color keyword.

In all syntax models, the keyword **color** preceding **COLOR** is only optional if **COLOR** is specified as a color keyword, without quotation marks.

Contents

Chapter 1. Installation	1-1
1.1 Operating Environment	1-2
1.1.1 Machine Requirements	1-2
1.1.2 Program Requirements	1-3
1.1.3 Serial Number	1-3
1.2 Installing EASEL OS/2 EE	1-4
1.2.1 Installing the EASEL OS/2 EE Development System	1-4
1.2.2 Installing Only the EASEL OS/2 EE Runtime Facility	1-5
1.2.3 Diskettes Required	1-6
1.3 Directories Created and Files Added	1-7
1.3.1 Directories Created During Installation	1-7
1.3.2 Files Added During Installation	1-7
1.4 Testing the System	1-12
1.4.1 Testing the EASEL OS/2 EE Runtime	1-12
1.4.2 Testing the EASEL OS/2 EE Compiler	1-12
1.5 Installation Error Messages	1-13
 Chapter 2. Welcome to EASEL OS/2 EE	 2-1
2.1 Why EASEL OS/2 EE?	2-2
2.1.1 Benefits of the EASEL OS/2 EE Language	2-3
2.2 Writing an Event-Driven Application	2-5
2.2.1 EASEL OS/2 EE Front-End Application	2-5
2.3 The Development Process	2-7
2.4 Distributing the Application	2-8
 Chapter 3. Programming with EASEL OS/2 EE	 3-1
3.1 Summary of Changes	3-2
3.1.1 New Features	3-2
3.1.2 Enhancements	3-3
3.2 How an Application is Created	3-5
3.3 Program Construction	3-6
3.3.1 Recommended Order of Program Components	3-6
3.3.2 Using Action Statements	3-7
3.4 Constructing a Program	3-8
3.4.1 An Introduction to Objects and Responses	3-8
3.4.2 Sample Program	3-9
3.4.3 Trying Out EASEL OS/2 EE	3-11
3.5 Creating, Compiling, and Executing an EASEL OS/2 EE Application	3-12
3.5.1 Creating an EASEL OS/2 EE Application	3-12
3.5.2 Compiling an EASEL OS/2 EE OS/2 EE Application	3-12
3.5.3 Executing an EASEL OS/2 EE Application	3-13
3.5.4 Editing the CONFIG.ESL File	3-13
3.6 How EASEL OS/2 EE Compiles and Executes the Program	3-15
3.7 Frame Oriented Syntax	3-16
3.7.1 Screen Size	3-16

3.7.2	Defining Frame Related Attributes	3-17
3.7.3	Changing Title Bar Text	3-18
3.8	Windowed versus Fullscreen Applications	3-19
3.8.1	Overview	3-19
3.8.2	Color Control	3-20
3.8.3	Pointing Devices	3-20
3.8.4	Scaling in a Windowed Application	3-20
3.8.5	Architectural Considerations in a Windowed Application	3-21
3.8.6	Sizing Considerations	3-21
3.8.7	System Menu Accelerators	3-21
3.8.8	Examples	3-21
Chapter 4.	Language Elements	4-1
4.1	The EASEL OS/2 EE Character Set	4-2
4.1.1	Blanks	4-2
4.1.2	Comments	4-3
4.2	National Language Support	4-4
4.2.1	Code Page Support	4-4
4.2.2	Keyboard Support	4-5
4.2.3	Date and Time Support	4-7
4.3	Keywords and Identifiers	4-8
4.4	Values	4-10
4.4.1	Categories and Types of Values	4-10
4.4.2	Compile-time Values	4-11
4.4.3	Literals	4-12
4.4.4	Constants	4-16
4.4.5	Variables	4-17
4.5	Assigning Values to Variables During Runtime	4-20
4.5.1	The copy Action Statement	4-20
4.5.2	The append Action Statement	4-21
4.5.3	The extract from Action Statement	4-22
4.6	Arrays: Defining and Referencing Groups of Constants or Variables	4-37
4.6.1	Specifying Dimensions in an Array Definition	4-37
4.6.2	Specifying Elements in an Array Definition	4-38
4.6.3	Global Variable Arrays	4-39
4.6.4	Referencing Elements in Arrays	4-39
4.6.5	Resizing an Array	4-40
4.7	Built-in Functions	4-42
4.7.1	Response Inquiry Functions	4-42
4.7.2	Object Inquiry Functions	4-43
4.7.3	Item Inquiry Functions	4-44
4.7.4	Action Inquiry Functions	4-44
4.7.5	Special Inquiry Functions	4-45
4.8	Expressions, Operators, and Operands	4-47
4.8.1	Arithmetic Operators	4-48
4.8.2	Relational Operators	4-50
4.8.3	Boolean Operators	4-51
4.9	Type Conversions	4-53
4.9.1	String/Integer and String/Floating Point Conversions	4-53

4.9.2 Boolean/String Conversions	4-54
4.9.3 Integer/Floating Point Conversions	4-54
4.9.4 String/Name Conversions	4-55
4.9.5 String/Name with Ancestry Conversions	4-55
4.10 Modules	4-57
Chapter 5. Objects, Coordinate Systems, and Fonts	5-1
5.1 Types of Objects and Their Uses	5-2
5.2 The Components of an Object	5-3
5.3 Common Object Attributes	5-5
5.3.1 Selectability	5-5
5.3.2 Visibility	5-8
5.3.3 Color	5-8
5.3.4 Ancestry	5-12
5.3.5 Priority	5-14
5.3.6 Parameter	5-15
5.4 Coordinate Systems and Cursors	5-16
5.4.1 The Screen Coordinate System	5-16
5.4.2 The Object Coordinate System	5-18
5.4.3 The Ancestor Coordinate System	5-20
5.4.4 Windows and Viewports: The Coordinate Systems of Regions	5-22
5.4.5 The Graphics and Text Cursors	5-32
5.5 Mixing Object Types in a Window	5-39
5.6 Fonts	5-42
5.6.1 Specifying a Font	5-42
5.6.2 Character Cells	5-43
5.6.3 Portability Across Monitors	5-44
5.6.4 EASEL OS/2 EE Fonts	5-45
5.6.5 Presentation Manager Fonts	5-47
5.6.6 Defining a Font	5-48
5.6.7 Image Fonts	5-49
5.6.8 Scalable Outline Fonts	5-50
5.6.9 Fonts in Textual Regions	5-50
5.7 Redrawing an Object	5-52
5.7.1 Redrawing a Specified Object	5-52
Chapter 6. Responses, Blocks, Action Statements, and Classes	6-1
6.1 About Responses	6-2
6.1.1 Overview of Responses and Their Uses	6-2
6.1.2 Responding to Program Startup	6-3
6.1.3 Responding to Objects	6-4
6.1.4 Responding to Action Bar Items	6-6
6.1.5 Using Response Inquiry Built-in Functions	6-7
6.2 Working with Blocks	6-10
6.2.1 Overview	6-10
6.2.2 Defining a Block	6-11
6.2.3 Nested Blocks	6-12
6.2.4 Leaving (Exiting) a Block	6-13
6.2.5 Responding to a Block Being Left Implicitly	6-16
6.2.6 Responding to Stimuli and Returning to a Specified Block	6-17

6.2.7	Disabling Responses Outside a Block	6-18
6.2.8	Responding to Specified Stimuli and Returning to a Block	6-19
6.2.9	Using the response to start Response Definition Within a Block	6-20
6.2.10	Using the response to char Response Definition in Blocks	6-21
6.3	About Action Statements	6-23
6.3.1	Overview of Actions and Their Uses	6-23
6.3.2	Specifying Action Statements	6-23
6.3.3	Creating, Changing, Clearing, and Deleting Objects	6-24
6.3.4	Changing Attributes of Objects	6-28
6.3.5	Changing the Size and Position of an Object	6-31
6.4	Using Classes of Objects	6-34
6.4.1	Overview	6-34
6.4.2	Defining a Class of Objects	6-34
6.4.3	Defining Responses for a Class of Objects	6-35
6.4.4	Specifying Actions for a Class of Objects	6-36
6.4.5	Adding an Object to a Class	6-37
6.4.6	Deleting an Object from a Class of Objects	6-38
6.4.7	Changing Class Membership	6-38
6.4.8	Using Built-in Functions with Classes	6-39
Chapter 7.	Graphical Objects, Image Regions, and Sense Regions	7-1
7.1	Graphical Objects	7-2
7.1.1	Defining Keys	7-2
7.1.2	Defining Graphical Regions	7-4
7.2	Drawing Statements for Graphical Objects	7-9
7.2.1	Overview	7-9
7.2.2	Placement of Drawing Statements	7-9
7.2.3	Moving the Cursors	7-10
7.2.4	Drawing a Line	7-11
7.2.5	Drawing a Box	7-13
7.2.6	Drawing a Polygon	7-14
7.2.7	Drawing a Circle	7-15
7.2.8	Drawing an Ellipse	7-16
7.2.9	Drawing a Wedge	7-18
7.2.10	Drawing Closed Figures (Shapes)	7-19
7.2.11	Defining and Referencing Patterns	7-30
7.2.12	Displaying Text in Graphical Objects	7-44
7.3	Action Statements for Changing Windows and Viewports in Graphical Regions	7-47
7.3.1	Introduction	7-47
7.3.2	Changing the Viewport Size	7-48
7.3.3	Changing the Viewport Position	7-50
7.3.4	Changing the Window Size	7-51
7.3.5	Changing the Window Position	7-52
7.4	Using Built-in Functions with Graphical Objects	7-54
7.4.1	Introduction	7-54
7.4.2	Using Functions to Inquire about Size and Position	7-54
7.4.3	Using Built-in Functions to Inquire about an Object's Attributes	7-59

7.5 Image Regions	7-63
7.5.1 Bitmap File Formats Supported	7-63
7.5.2 Operations Supported	7-64
7.5.3 Defining Image Regions	7-64
7.5.4 Foreground and Background Colors	7-65
7.5.5 Cache Specifications	7-65
7.5.6 Scaling	7-67
7.5.7 Specifying File Contents	7-68
7.5.8 Specifying Image Maps in Image Regions	7-69
7.5.9 Read/Write Statements	7-70
7.6 Sense Regions	7-72
7.6.1 Defining Sense Regions	7-72
7.6.2 The xcoord and ycoord Built-in Functions	7-73
7.6.3 PM Frame Attributes	7-73
Chapter 8. Textual Regions	8-1
8.1 Defining and Using Textual Regions	8-2
8.1.1 Overview	8-2
8.1.2 Defining a Textual Region	8-3
8.2 Drawing Statements for Textual Regions	8-7
8.2.1 Overview	8-7
8.2.2 Moving the Text Cursor	8-9
8.2.3 Defining Text Segments and Text Blocks	8-10
8.2.4 Inserting Text	8-14
8.2.5 Overwriting Text	8-20
8.3 Response Definitions for Textual Regions	8-26
8.4 Action Statements for Textual Regions	8-27
8.4.1 Overview	8-27
8.4.2 Adding Drawing Statements to a Textual Region	8-28
8.4.3 Removing Text	8-28
8.4.4 Controlling Text Emphasis	8-29
8.4.5 Changing the Visibility of the Text Cursor	8-30
8.4.6 Writing a Textual Region to a File	8-30
8.4.7 Reading a File into a Textual Region	8-32
8.4.8 Finding a String of Text in a Textual Region	8-34
8.4.9 Changing the Size and Position of a Textual Region's Window and Viewport	8-36
8.5 Using Built-in Functions with Textual Regions	8-41
8.5.1 Overview	8-41
8.5.2 Using the textual Built-in Functions	8-43
8.5.3 Using the column size of Built-in Function	8-46
8.5.4 Using the line size of Built-in Function	8-47
8.5.5 Using the foreground at and background at Built-in Functions	8-47
8.5.6 Using the found Built-in Function	8-48
8.6 Sample Programs Using Textual Regions	8-49
8.6.1 Sample #1	8-49
8.6.2 Sample #2	8-50
8.7 Working with Colored Textual Regions	8-54
8.7.1 Defining a Colored Textual Region	8-54
8.7.2 Coloring Text That Already Exists in the Program	8-55

8.7.3 Specifying Colors for Text in a File: Using the CText Module 8-57

Chapter 9. Dialog Boxes, Dialog Regions, and Dialog Controls	9-1
9.1 Dialog Boxes	9-2
9.1.1 Defining Dialog Boxes	9-2
9.1.2 Attributes for Dialog Boxes	9-3
9.1.3 Actions for Dialog Boxes	9-3
9.1.4 Keyboard and Mouse Use in Dialog Boxes	9-3
9.1.5 Example of a Dialog Box	9-4
9.1.6 Built-in Functions for Dialog Boxes	9-4
9.2 Dialog Regions	9-5
9.2.1 Actions for Dialog Regions	9-6
9.2.2 Keyboard and Mouse Use in Dialog Regions	9-6
9.2.3 Built-in Functions for Dialog Regions	9-6
9.3 Dialog Control Objects	9-7
9.3.1 Overview	9-7
9.3.2 Groups	9-7
9.3.3 Mnemonics	9-7
9.3.4 Built-in Functions for Dialog Controls	9-8
9.3.5 Single-Line Entry Fields	9-8
9.3.6 Multiple-Line Entry Fields	9-9
9.3.7 List Boxes	9-11
9.3.8 Combination Boxes	9-15
9.3.9 Drop-down Lists	9-16
9.3.10 Push Buttons	9-17
9.3.11 Radio Buttons	9-18
9.3.12 Check Boxes	9-20
9.3.13 Static Text	9-21
9.3.14 Group Boxes	9-23
9.4 Sample Code for Generating a Dialog Box	9-24
Chapter 10. Region Attributes and Action Bars	10-1
10.1 Region Attributes	10-2
10.1.1 Using the Border Attribute	10-4
10.1.2 Using the Title Bar Attribute	10-4
10.1.3 Using the Minimize Attribute	10-4
10.1.4 Using the Maximize Attribute	10-5
10.1.5 Using the System Menu Attribute	10-5
10.1.6 Using the Scroll Attributes	10-6
10.1.7 Using the Action Bar Attribute	10-6
10.1.8 Using the No Scale Attribute	10-7
10.1.9 Using the Pointer Attribute	10-7
10.2 Region Actions	10-8
10.3 Creating Action Bars	10-9
10.3.1 Defining an Action Bar Template	10-10
10.3.2 Defining Pulldowns	10-11
10.3.3 Defining Choices	10-12
10.3.4 Defining Separators	10-13
10.3.5 Defining Buttons	10-13
10.3.6 Defining Global Items	10-14

10.3.7 Accelerator Keys	10-14
10.3.8 Mnemonics	10-15
10.3.9 Item Parameters	10-16
10.3.10 An Example Action Bar Template	10-17
10.4 Manipulating Action Bars and Items	10-19
10.4.1 Manipulating Action Bars	10-19
10.4.2 Manipulating Action Bar Items	10-20
10.5 Responding to Action Bar Items	10-27
10.5.1 Responding to Items in a Region	10-27
10.6 Built-in Functions for Action Bar Items	10-28
10.6.1 Built-in Functions in Inquiries	10-28
10.6.2 Built-in Functions in Responses	10-29
Chapter 11. Program Flow and Performance	11-1
11.1 Controlling Program Flow	11-2
11.2 Functions, Subroutines, and Action Routines	11-4
11.2.1 Declaring External Subroutines and External Functions	11-4
11.2.2 Action Routines and Internal EASEL OS/2 EE Subroutines	11-4
11.3 General Program Control	11-8
11.3.1 Including Another EASEL OS/2 EE File in the Current Program	11-8
11.3.2 Exiting the Program	11-9
11.3.3 Invoke	11-10
11.3.4 Transferring to Another Program	11-11
11.3.5 Saving a Program in Progress	11-12
11.4 Global and Local Entities	11-14
11.4.1 Variables	11-14
11.4.2 Applications	11-14
11.4.3 Devices	11-14
11.4.4 Memory	11-15
11.4.5 Global Entities in Changed Programs	11-15
11.4.6 Global and Local Values in Saved Programs	11-15
11.5 Controlling Flow Within a Program	11-17
11.5.1 Referencing an Action Routine	11-17
11.5.2 Specifying a Conditional Action Statement	11-18
11.5.3 Specifying a Call Action Statement	11-21
11.5.4 Specifying a for or while loop	11-21
11.5.5 for each member loop	11-23
11.5.6 for each selected line loop	11-24
11.5.7 Exiting from a Loop	11-24
11.5.8 Nested Loops	11-25
11.5.9 Blocks and Loops	11-26
11.5.10 Switch Statement	11-26
11.6 Program Timing	11-29
11.6.1 Responding to Elapsed Time	11-29
11.6.2 Responding to Elapsed Time During Which No Stimulus is Received	11-30
11.6.3 Action Statement to Suspend Processing	11-31
11.7 Runtime Memory Handling	11-33
11.7.1 Global Memory	11-33

11.7.2	Changing Memory Allocation in the EASEL.CMD File	11-34
11.7.3	Controlling Memory from within the Program	11-34
Chapter 12.	Layout/CUA	12-1
12.1	Before You Begin	12-2
12.1.1	What You Need to Use Layout/CUA	12-2
12.1.2	What You Need to Know	12-2
12.1.3	About CUA Conformance	12-2
12.1.4	About This Chapter	12-3
12.2	Using Layout/CUA	12-4
12.2.1	The Layout/CUA Primary Window	12-4
12.2.2	Using the Help System	12-5
12.2.3	Creating an Interface Primary Window	12-7
12.2.4	Creating Secondary Windows	12-8
12.2.5	Creating Dialog Boxes	12-8
12.2.6	Selecting a Tool from the Palette	12-9
12.2.7	Adding Controls to a Dialog Box or Dialog Region	12-11
12.2.8	Selecting Controls Inside a Dialog Box or Dialog Region	12-11
12.2.9	Deleting Objects	12-12
12.2.10	Moving Objects	12-12
12.2.11	Resizing Objects and Groups	12-12
12.2.12	Aligning Objects	12-13
12.2.13	Modifying Object Information	12-13
12.2.14	Editing the Action Bar	12-14
12.2.15	Editing a Pulldown	12-14
12.2.16	Undoing Your Last Action	12-14
12.2.17	Hiding and Showing Windows	12-14
12.2.18	Creating Selection Fields	12-15
12.2.19	Using the Clipboard	12-16
12.2.20	Converting the Interface to EASEL OS/2 EE Code	12-17
12.2.21	Saving the Interface	12-19
12.2.22	Editing an Interface File	12-20
12.3	Code Generation	12-21
12.3.1	Dialog Box Subroutines	12-21
12.3.2	Automatic Dialog-Box Calling	12-23
12.3.3	Pulldown Choices	12-23
12.3.4	Push Buttons	12-23
12.4	Object Information	12-24
12.4.1	Primary Window Info	12-24
12.4.2	Action Bar Choice Info	12-25
12.4.3	Pulldown Choice Info	12-25
12.4.4	Secondary Window Info	12-27
12.4.5	Dialog Box Info	12-28
12.4.6	Push Button Info	12-29
12.4.7	Radio Button Info	12-30
12.4.8	Check Box Info	12-31
12.4.9	List Box Info	12-32
12.4.10	Entry Field Info	12-33
12.4.11	Static Text Info	12-34
12.4.12	Group Box Info	12-34

12.4.13 Multiline Entry Field Info	12-35
12.4.14 Combination Box Info	12-36
Chapter 13. Library Subroutines and Functions	13-1
13.1 Overview	13-2
13.1.1 Include File	13-2
13.1.2 Library Subroutines	13-2
13.1.3 Library Functions	13-3
13.2 The EASEL OS/2 EE Library Functions	13-4
13.2.1 EslQueryEgaColor() Function	13-4
13.2.2 EslSetEgaColor() Function	13-4
13.2.3 EslQueryFocus() Function	13-5
13.3 The Date Library Functions	13-7
13.3.1 DaySpan() Function	13-7
13.3.2 LocalDate() Function	13-8
13.3.3 LocalTime() Function	13-8
13.3.4 Valldate() Function	13-9
13.3.5 WeekDay() Function	13-10
13.4 The Math Library Functions and Subroutines	13-11
13.4.1 The Math Library Functions	13-11
13.4.2 The Math Library Subroutines	13-12
13.4.3 Error Handling	13-13
13.4.4 ABS() Function	13-14
13.4.5 ARCCOS() Function	13-14
13.4.6 ARCSIN() Function	13-15
13.4.7 ARCTAN() Function	13-15
13.4.8 ARCTAN2() Function	13-15
13.4.9 CEIL() Function	13-16
13.4.10 COS() Function	13-16
13.4.11 E() Function	13-17
13.4.12 EXP() Function	13-17
13.4.13 FLOOR() Function	13-17
13.4.14 LN() Function	13-18
13.4.15 LOG() Function	13-18
13.4.16 MOD() Function	13-18
13.4.17 PI() Function	13-19
13.4.18 POWER() Function	13-19
13.4.19 SIN() Function	13-20
13.4.20 SQRT() Function	13-20
13.4.21 TAN() Function	13-20
13.4.22 MAX() Subroutine	13-21
13.4.23 MAXINT() Subroutine	13-21
13.4.24 MEAN() Subroutine	13-21
13.4.25 MEANINT() Subroutine	13-22
13.4.26 MEDIAN() Subroutine	13-22
13.4.27 MEDIANINT() Subroutine	13-22
13.4.28 MIN() Subroutine	13-23
13.4.29 MININT() Subroutine	13-23
13.4.30 STDDEV() Subroutine	13-23
13.4.31 STDDEVINT() Subroutine	13-24

13.4.32	SUM() Subroutine	13-24
13.4.33	SUMINT() Subroutine	13-24
13.4.34	VARIANCE() Subroutine	13-25
13.4.35	VARIANCEINT() Subroutine	13-25
13.5	The File I/O Library	13-26
13.5.1	Overview	13-26
13.5.2	The File I/O Library Subroutines	13-26
13.5.3	The Record Definition	13-27
13.5.4	The Open() and Close() Subroutines	13-31
13.5.5	Input Subroutines	13-35
13.5.6	Output Subroutines	13-41
13.5.7	Parameter Adjustment Subroutines	13-44
13.5.8	Error Handling Subroutine	13-47
13.6	The Message Library Function	13-49
Chapter 14.	Application Communications, PM Clipboard, and Stimulus	14-1
14.1	Overview of Applications	14-2
14.1.1	Application Types: Remote and Local	14-2
14.1.2	Defining, Starting, and Stopping Applications	14-3
14.2	Sending and Receiving Data	14-7
14.2.1	Sending Information	14-7
14.2.2	Responding to Data Received from an Application	14-7
14.2.3	How response to line and response to char Are Processed	14-13
14.2.4	Changing the Disposition of Text Input	14-16
14.3	Remote Application Support for EASEL OS/2 EE	14-18
14.3.1	Starting Remote Applications	14-18
14.3.2	Connection Types	14-18
14.3.3	Establishing a Connection	14-20
14.3.4	ACDI Remote Characteristics Keywords	14-21
14.3.5	Failure Codes	14-24
14.3.6	Errorlevel Built-in Function	14-24
14.4	Sample Programs	14-26
14.4.1	Sample Program Using a Local and Remote Application	14-26
14.4.2	Sample Program Using Characters Received from the Keyboard	14-27
14.5	Using the Presentation Manager Clipboard	14-28
14.6	Stimulus Libraries	14-29
14.6.1	Declaring a Stimulus	14-29
14.6.2	Defining a Stimulus Response	14-29
14.6.3	Response Inquiry Functions	14-30
Chapter 15.	Controlling the Keyboard and Pointing Devices	15-1
15.1	About Pointing Devices	15-2
15.1.1	Mouse	15-2
15.1.2	Touchscreen	15-2
15.1.3	Mouse Pointer Appearance	15-2
15.1.4	Touchscreen Support	15-3
15.2	Controlling Availability of Pointing Devices	15-6
15.2.1	Using the Default Pointing Device	15-6
15.3	Opening and Closing a Pointing Device	15-7

15.3.1 Controlling Input from a Pointing Device	15-7
15.4 Handling Characters Received from the Keyboard	15-9
15.4.1 Matching Special Characters from the Keyboard	15-9
15.5 Providing Feedback to Selections	15-14
15.5.1 Providing Audio Feedback to Selections	15-14
15.5.2 Providing Video Feedback to Selections	15-14
Chapter 16. Business Graphics	16-1
16.1 Introduction	16-2
16.2 Creating a Graph	16-3
16.2.1 Required Procedures	16-3
16.2.2 Action Routines for Each Graph Type	16-5
16.2.3 Specifying Data Elements for a Graph	16-6
16.2.4 Specifying the Number of Data Sets and Data Elements in a Graph	16-8
16.2.5 Specifying the Decimal Point Character	16-8
16.3 Rules and Variables for Certain Graph Types	16-10
16.3.1 Overlapping Bar Graph	16-10
16.3.2 Three-Dimensional (3-D) Bar Graph	16-10
16.3.3 3-D Outline Bar Graph	16-11
16.3.4 Pie Graph	16-11
16.3.5 Stepped Line Graph	16-13
16.3.6 Stacked Line Graph	16-13
16.4 Customizing a Graph	16-14
16.4.1 Graph Titles	16-15
16.4.2 Axis Titles	16-16
16.4.3 Axis Boundaries	16-19
16.4.4 Tick and Pie Wedge Labels	16-20
16.4.5 Legends	16-24
16.4.6 Footnotes	16-26
16.4.7 Colors	16-27
16.4.8 Coloring Text in a Graph (Reference)	16-29
16.4.9 Grid Lines	16-29
16.4.10 Marker Lines	16-29
16.4.11 Line Widths	16-32
16.4.12 Symbols	16-33
16.4.13 Superimposing Two Graphs	16-34
16.4.14 GMovables: Moving Text in a Graph	16-34
16.5 Sample Program Using the BGraph Module	16-37
16.6 Illustrations of Graph Types	16-40
16.7 Summary of BGraph Module Variables	16-47
16.8 Public File of the BGRAPH Module	16-50
Chapter 17. SQL Support	17-1
17.1 Introduction	17-2
17.1.1 Requirements	17-2
17.1.2 Access Plans	17-2
17.1.3 Types of Commands	17-2
17.1.4 Input and Output	17-2
17.2 Direct Commands	17-3

17.2.1	startdb Direct Command	17-4
17.2.2	stopdb Direct Command	17-5
17.2.3	binddb Direct Command	17-6
17.3	Configuration Commands	17-7
17.3.1	prompt Configuration Command	17-8
17.3.2	error Configuration Command	17-9
17.3.3	null Configuration Command	17-10
17.3.4	info Configuration Command	17-11
17.3.5	bindfile Configuration Command	17-12
17.3.6	output Configuration Command	17-13
17.3.7	fs Configuration Command	17-14
17.3.8	rs Configuration Command	17-15
17.3.9	set Configuration Command	17-16
17.3.10	maxwidth Configuration Command	17-17
17.3.11	startq Configuration Command	17-18
17.3.12	verbose Configuration Command	17-19
17.3.13	exit Configuration Command	17-20
17.4	Inquiry Commands	17-21
17.4.1	display Inquiry Command	17-22
17.4.2	names Inquiry Command	17-23
17.4.3	types Inquiry Command	17-24
17.4.4	widths Inquiry Command	17-25
17.5	Dynamic SQL Commands	17-26
17.6	Error Handling	17-27
17.6.1	Error Detected by Local Application	17-27
17.6.2	Error Detected by OS/2 Extended Edition Database Manager	17-27
17.7	Command Line Options	17-30
17.8	Notes on Using SQL Support	17-31
17.8.1	Customizing Strings	17-31
17.8.2	Using Output as an Input Command	17-31
17.8.3	LONG VARCHAR Allocation	17-31
17.8.4	Increasing Memory	17-32
17.8.5	Requirement for Dynamic Commands	17-32
17.8.6	Statement Size	17-32
17.8.7	Case and White Space	17-32
17.8.8	Shared Mode	17-32
17.8.9	Isolation Level	17-32
17.8.10	Standard Output and Standard Error	17-32
17.8.11	Determining Number of Records	17-32
17.8.12	Error Message	17-32
17.8.13	Graphic Data Types	17-33
Chapter 18.	Audio Support	18-1
18.1	Overview	18-2
18.1.1	Digitized Audio Files	18-2
18.1.2	Audio Utility	18-3
18.1.3	Include File	18-3
18.1.4	Audio Dynamic Link Library	18-4
18.1.5	Responding to Audio Events	18-4

18.2	EASEL OS/2 EE Audio Support Subroutines	18-6
18.2.1	Setting Argument Values	18-7
18.2.2	Return Codes	18-7
18.2.3	AUDClose()	18-9
18.2.4	AUDErase()	18-10
18.2.5	AUDOpen()	18-11
18.2.6	AUDPause()	18-12
18.2.7	AUDPlay()	18-13
18.2.8	AUDQueryBusy()	18-17
18.2.9	AUDQueryElapsedTime()	18-19
18.2.10	AUDQueryPlayingTime()	18-20
18.2.11	AUDRecord()	18-21
18.2.12	AUDResume()	18-25
18.2.13	AUDSetBalance()	18-26
18.2.14	AUDSetVolume()	18-28
18.2.15	AUDStop()	18-29
18.3	Sample Program Using Audio Support	18-31
18.4	Audio Support Reserved Constants	18-33
Chapter 19.	APPC Support	19-1
19.1	Overview	19-2
19.1.1	The Automatic Interface	19-3
19.1.2	The Controlled Interface	19-4
19.1.3	Responding to APPC Events	19-4
19.1.4	APPC Dynamic Link Library	19-5
19.1.5	Include File	19-5
19.1.6	Confirmation Processing	19-5
19.1.7	Forcing a Send	19-6
19.1.8	How EASEL OS/2 EE APPC Support Works	19-6
19.2	EASEL OS/2 EE APPC Events	19-8
19.3	EASEL OS/2 EE APPC Subroutines	19-11
19.3.1	Setting/Querying Profile Information	19-12
19.3.2	The Automatic Interface	19-14
19.3.3	Sending/Receiving Non-string Data	19-15
19.3.4	Confirmation Processing	19-16
19.3.5	The Controlled Interface	19-18
19.3.6	Utility Subroutines	19-22
19.4	Using APPC Subroutines	19-23
19.4.1	Setting Argument Values	19-23
19.4.2	Return Codes	19-24
19.5	Detailed Subroutine Descriptions	19-26
19.5.1	APPCAcceptStart()	19-26
19.5.2	APPCConvertA2E()	19-28
19.5.3	APPCConvertE2A()	19-31
19.5.4	APPCEnd()	19-34
19.5.5	APPCFlush()	19-36
19.5.6	APPCGetAcceptProfile()	19-38
19.5.7	APPCGetAcceptProfileNames()	19-41
19.5.8	APPCGetDLLData()	19-43
19.5.9	APPCGetInfo()	19-46

19.5.10	APPCGetInitProfile()	19-50
19.5.11	APPCGetInitProfileNames()	19-53
19.5.12	APPCGetString()	19-55
19.5.13	APPCGetSystemError()	19-57
19.5.14	APPCReadyToReceive()	19-61
19.5.15	APPCReceiveNotify()	19-63
19.5.16	APPCRequestConfirm()	19-65
19.5.17	APPCRequestToSend()	19-67
19.5.18	APPCSendConfirmed()	19-69
19.5.19	APPCSendDLLData()	19-71
19.5.20	APPCSendError()	19-75
19.5.21	APPCSendString()	19-77
19.5.22	APPCSetAcceptProfile()	19-81
19.5.23	APPCSetInitProfile()	19-84
19.5.24	APPCStart()	19-88
19.5.25	APPCTestReqToSend()	19-91
19.6	Sample Code: Automatic Interface	19-93
19.7	Sample Code: Controlled Interface	19-95
19.8	EASEL OS/2 EE APPC Reserved Constants	19-98
19.9	Communications Manager Considerations	19-100
19.9.1	Configuring CM for an EASEL OS/2 EE Acceptor	19-100
19.9.2	Enabling APPC Tracing	19-101
Chapter 20.	3270 Support	20-1
20.1	Introduction	20-2
20.1.1	EASEL OS/2 EE in the 3270 Environment	20-2
20.1.2	Components of 3270 Support for EASEL OS/2 EE	20-2
20.1.3	Using the M3270 Module: Required Procedures	20-3
20.1.4	Referencing the M3270 Module	20-3
20.1.5	Sample Program	20-4
20.2	3270 Session and Screen Concepts	20-5
20.2.1	Logical Terminal Sessions	20-5
20.2.2	Screen Sizes	20-6
20.2.3	Field-oriented and Line-oriented Screens	20-6
20.3	Retrieving Field Information with FIELDS.EXE	20-8
20.3.1	FIELDS.EXE Sample Output	20-10
20.4	M3270 Module Action Routines	20-13
20.4.1	M3270 Action Routines and EASEL OS/2 EE Built-in Functions	20-13
20.4.2	Summary of M3270 Action Routines	20-14
20.4.3	Starting and Stopping 3270 Support for EASEL OS/2 EE	20-17
20.4.4	Connecting and Disconnecting Sessions	20-19
20.4.5	Examining the Buffer	20-21
20.4.6	Synchronizing EASEL OS/2 EE with the Host	20-33
20.4.7	Communicating with the Host	20-45
20.4.8	Controlling the Emulator Window	20-50
20.4.9	Miscellaneous Action Routines	20-56
20.5	Error Handling	20-64
20.5.1	Error Status Variables	20-64
20.5.2	Error Messages (Cross Reference)	20-66

20.6	Reserved Variables (Cross Reference)	20-71
20.7	Reserved Constants	20-74
20.8	Sample Program Using 3270 Screens	20-75
20.8.1	Sample Program Using EASEL OS/2 EE 3270 Support	20-77
Chapter 21.	5250 Support	21-1
21.1	Introduction	21-2
21.1.1	EASEL OS/2 EE in the 5250 Environment	21-2
21.1.2	Components of 5250 Support for EASEL OS/2 EE	21-2
21.1.3	Using the M5250 Module: Required Procedures	21-3
21.1.4	Referencing the M5250 Module	21-3
21.1.5	Sample Program	21-4
21.2	5250 Session and Screen Concepts	21-5
21.2.1	Session Names	21-5
21.2.2	Session States	21-5
21.2.3	Field-oriented and Line-oriented Screens	21-6
21.3	Retrieving Field Information with FIELDS52.EXE	21-9
21.3.1	Reading the FIELDS52.EXE Report	21-9
21.3.2	Using FIELDS52.EXE	21-9
21.3.3	FIELDS52.EXE Sample Output	21-11
21.4	M5250 Module Action Routines	21-13
21.4.1	M5250 Action Routines and EASEL OS/2 EE Built-in Functions	21-13
21.4.2	Summary of M5250 Action Routines	21-14
21.4.3	Starting and Stopping 5250 Support for EASEL OS/2 EE	21-17
21.4.4	Connecting and Disconnecting Sessions	21-19
21.4.5	Examining the Buffer	21-21
21.4.6	Synchronizing 5250 Support for EASEL OS/2 EE with the Host	21-33
21.4.7	Synchronizing with AS/400 Applications	21-34
21.4.8	Synchronizing with Applications Via 3270 Device Emulation	21-37
21.4.9	Communicating with the Host	21-49
21.4.10	Miscellaneous Action Routines	21-55
21.5	Error Handling	21-60
21.5.1	Error Status Variables	21-60
21.5.2	Error Messages (Cross Reference)	21-62
21.6	Reserved Variables (Cross-Reference)	21-67
21.7	Reserved Constants	21-70
21.8	Sample Program	21-71
21.8.1	Sample Program Using 5250 Support for EASEL OS/2 EE	21-73
Chapter 22.	Printing and Plotting	22-1
22.1	Printing and Plotting EASEL OS/2 EE Regions	22-2
22.1.1	About Output Devices	22-2
22.1.2	Overview of Hardcopy Support	22-2
22.1.3	Using the Plot Action Statement	22-3
22.2	Printing ASCII Files Within EASEL OS/2 EE	22-5
22.2.1	Using the Printing Utilities	22-5
22.2.2	Optional Parameters	22-5
22.2.3	Example	22-6

22.2.4 Error Messages	22-7
Chapter 23. Environment Files	23-1
23.1 Overview	23-2
23.2 COMPILE.CMD OS/2 Command File	23-3
23.2.1 Command Line Options	23-3
23.3 CONFIG.ESL File	23-5
23.3.1 Configuration Options	23-5
23.4 EASEL.CMD OS/2 Command File	23-9
23.4.1 Command Line Options	23-9
Chapter 24. Debugging	24-1
24.1 Overview	24-2
24.1.1 Using the Errorlog File	24-2
24.1.2 Reading Errors on Another Terminal	24-2
24.2 Action Statements and Environment Declarations for Debugging	24-3
24.2.1 Tracing Action Statements for Portions of an EASEL OS/2 EE Program	24-3
24.2.2 Sending Values to the Errorlog	24-3
24.3 Command Line Options for Debugging	24-5
24.3.1 Tracing Applications Communications	24-5
24.3.2 Tracing Action Statements for an Entire EASEL OS/2 EE Program	24-5
Chapter 25. Writing Local Applications and Libraries	25-1
25.1 Writing Well-behaved Local Applications	25-2
25.1.1 Standalone Program	25-2
25.1.2 Keyboard and Screen I/O	25-2
25.1.3 Guidelines for Writing C Local Applications for EASEL OS/2 EE	25-2
25.2 Writing Library Functions and Subroutines	25-3
25.2.1 Declaring C Routines in an EASEL OS/2 EE Program	25-3
25.2.2 Function Arguments and Return Values	25-4
25.2.3 Subroutine Arguments and Return Values	25-4
25.2.4 Strings	25-5
25.2.5 C Coding Considerations	25-7
25.2.6 Example Library and Usage	25-8
25.3 Writing a Stimulus Library	25-18
Appendix A. Include Files	A-1
A.1 Overview	A-2
A.2 accel.inc file	A-3
A.3 datelib.inc file	A-4
A.4 egacolor.inc File	A-5
A.5 eslappc.inc file	A-6
A.6 eslaudio.inc file	A-12
A.7 esllib.inc file	A-14
A.8 fileio.inc File	A-15
A.9 mathlib.inc File	A-16
A.10 message.inc File	A-18

A.11 pmptr.inc File	A-20
Appendix B. Specified Operating Environment	B-1
B.1 Machine Requirements	B-2
B.2 Program Requirements	B-4
B.3 Graphics Display Considerations	B-5
B.3.1 Display Resolution	B-5
B.3.2 Portability	B-5
Appendix C. Fatal Runtime Errors	C-1
C.1 Overview	C-2
C.2 Error Messages	C-3
C.2.1 ESL1002	C-3
C.2.2 ESL1003	C-3
C.2.3 ESL1004	C-3
C.2.4 ESL1005	C-3
C.2.5 ESL1006	C-4
C.2.6 ESL1007	C-4
C.2.7 ESL1008	C-4
C.2.8 ESL1009	C-4
C.2.9 ESL1010	C-5
C.2.10 ESL1011	C-5
C.2.11 ESL1012	C-5
C.2.12 ESL1012	C-5
C.2.13 ESL1012	C-5
Index	X-1

Chapter 16. Business Graphics

16.1 Introduction

The EASEL OS/2 EE Business Graphics module, called the **BGraph** module, is comprised of a set of predefined variables and action subroutines that enable you to easily convert tables of numbers into several different kinds of graphs:

- LINE GRAPHS
 - Stacked line graph
 - Standard line graph
 - Standard line graph with symbols
 - Stepped line graph
 - Surface line graph
- HI-LO-CLOSE GRAPH
- PIE GRAPH
- BAR GRAPHS
 - Standard bar graph
 - 3-D bar graph
 - Overlapping bar graph
 - 3-D overlapping bar graph
 - Stacked bar graph
 - 3-D stacked bar graph
- SCATTER GRAPH

Your tasks are to specify the region where the graph will be placed, identify the data to be plotted, and select the type of graph to be displayed. The module draws the graph.

You can also specify:

- A title for the graph and axes
- Automatic or manual axis boundaries
- Labels for ticks and pie wedges
- Colors to be used for text, bars, lines, and pie wedges
- Footnotes
- Legends
- Axis grid lines
- Fonts and colors for titles, labels, footnotes, and legends
- Manually controlled tick marks
- Line widths
- Patterns for symbols

16.2 Creating a Graph

16.2.1 Required Procedures

The following instructions explain how to use the **BGraph** module to create a graph.

Note: The predefined variables and action subroutines must be specified *exactly* as shown in boldface type.

1. Reference the **BGraph** module by inserting the following line at the top of your program:

```
module BGraph
```

All the variables and action subroutines are now available for the program.

2. Using EASEL OS/2 EE syntax, create a region for the graph to be drawn in. For example:

```
region SalesGraph size 300 300  
    at position 100 50 in desktop blue border
```

3. Use the **copy to GRegion** command to copy the name of the region into the string variable **GRegion**. The following model gives the syntax for the command:

Action: Specifies the name of the region to contain the graph.

Model

copy GRAPH_REGION to GRegion

Where: GRAPH_REGION is the name of the region you created in step 2 above.

Example: copy "SalesGraph" to GRegion

- Specify the data elements for the graph. Use the **copy to GDataY** command to copy your input data into the **GDataY** string array variable. For example, if the input data came from the following table:

SALES VOLUME (in thousands)

	'82	'83	'84	'85
Product A	24	29	15	27
Product B	34	45.5	53	61
Product C	18	33	22	42

The *data elements* (24, 29, 15, 27, etc.) are grouped by *data sets* (Product A, Product B, and Product C). You reference a data element by specifying the data set and then the associated data element. For example:

```
copy "24" to GDataY[1,1]      # This is the first data
                               # point in data set 1.

copy "29" to GDataY[1,2]
copy "15" to GDataY[1,3]
copy "27" to GDataY[1,4]
copy "34" to GDataY[2,1]
copy "45.5" to GDataY[2,2]
copy "53" to GDataY[2,3]
copy "61" to GDataY[2,4]
copy "18" to GDataY[3,1]
copy "33" to GDataY[3,2]
copy "22" to GDataY[3,3]
copy "42" to GDataY[3,4]      # This is the fourth data
                               # point in data set 3.
```

- Copy the number of data sets and elements per data set in the current graph to the variables **GDataSets** and **GDataElements**, respectively. For example, using the sample input data from step 4 above:

```
copy 3 to GDataSets
copy 4 to GDataElements
```

If the number of data sets is greater than 10, you must change the value of the constant **GMaxDataSets**. If the number of data elements per data set is greater than 20, you must change the value of the constant **GMaxDataElements**. (Set these constants as low as possible to avoid allocating unnecessary memory.)

- If you choose to set optional variables for the graph, such as titles and fonts, legend, colors, and grid lines, refer to 16.4, "Customizing a Graph" on page 16-14 for descriptions of these options before continuing.
- Specify the action statement for the type of graph. For certain graph types, you must also set required variables. Find the specific type of graph you want to create in 16.2.2, "Action Routines for Each Graph"

Type" on page 16-5, and follow the procedures specific to that graph type.

8. When a graph is created, the region specified by the variable **GRegion** is cleared, and any children of this region are deleted. Because of this, if you want other objects in the region, you must add them to the region *after* the graph is drawn.

16.2.2 Action Routines for Each Graph Type

<i>To draw the graph type...</i>	<i>Specify...</i>	Notes
Standard Line	action GLines	
Line with Symbols	action GLinesWithSymbols	
Stacked Line	action GStackedLines	
Surface Line	action GSurfaceLines	The data sets are graphed in the order in which they are placed into GDataY .
Stepped Line	copy true to GStepLineSet	[DATA_SET_NUMBER]
	action GLines	
Hi-Lo-Close	action GHiLoClose	The first data set is "High," the second is "Low," and the third is "Close."
Pie	action GPie	GPieDataSet specifies the data set to use to draw the Pie. GraphPie indicates whether a Pie was drawn with the given input data. A pie graph cannot be created from mixed positive and negative data.
Standard Bar	action GBars	
3-D Bar	copy true to Graph3D	
	action GBars	
Overlapping Bar	copy true to GraphOverlap	
	action GBars	

3-D Overlapping Bar	copy true to Graph3D
	copy true to GraphOverlap
	action GBars
Stacked Bar	action GStackedBars
3-D Stacked Bar	copy true to Graph3D
	action GStackedBars
Scatter Graph	action GScatter

16.2.3 Specifying Data Elements for a Graph

GDataY and **GDataX** are string arrays that hold the positions of the data elements of the graph. **GDataY** is required for all graphs; **GDataX** is required only for a graph that has X-axis data (Line, Line with Symbols, or Scatter graphs).

Action: Assigns the data element of the graph.

Model (For all graphs)

```
copy { DATA_ELEMENT_VALUE | GMissingDataValue } to
      GDataY[ DATA_SET_INDEX, DATA_ELEMENT_INDEX ]
```

Model (For Line, Line with Symbols, or Scatter Graphs)

```
copy DATA_ELEMENT_VALUE to
  { GDataY[ DATA_SET_INDEX, DATA_ELEMENT_INDEX ] |
    GDataX[ DATA_SET_INDEX, DATA_ELEMENT_INDEX ] }
```

Where: DATA_ELEMENT_VALUE is a value that will be plotted on the graph. Because **GDataY** and **GDataX** are string arrays, DATA_ELEMENT_VALUE can include commas, decimal points, dollar signs, fractions (including those such as “12/7” or “5 8/7”), and parentheses (which indicate a negative value).

DATA_SET_INDEX is an integer representing the first array subscript for the position of the DATA_ELEMENT_VALUE in the array.

DATA_ELEMENT_INDEX is an integer representing the second array subscript for the position of the **DATA_ELEMENT_VALUE** in the array.

For example: in **GDataY[3,2]**, 3 is the **DATA_SET_INDEX** and 2 is the **DATA_ELEMENT_INDEX**

Examples: copy "5 3/4" to **GDataY[1,1]**
copy ".0357" to **GDataX[3,4]**
copy "\$5,000,000,000.03" to **GDataY[10,12]**
copy "(10)" to **GDataY[5,1]**

The **BGraph** module has an array for each set of axis coordinates: **GDataY** for the Y coordinates, and **GDataX** for the X coordinates. If you are producing a graph with X-axis data, you must copy values into both arrays.

When automatic scaling is used, EASEL OS/2 EE takes only the first five digits of an integer or large decimal number (such as 12345.10) and the first five digits to the right of the decimal point in a small number (such as 0.11112). In the following example, all the numbers will be truncated to 11,111:

111,111 111,112 111,113

In this case, you should use manual scaling, which takes any number of digits.

GMissingDataValue allows you to specify a "blank" data point in the Y axis of a Line or Scatter graph. By default, a missing data element in the **GDataY** array is plotted as a "0" value, and, in a Line graph, the line is drawn to that point. To avoid the line being drawn to the "0," you can copy **GMissingDataValue** to the element's position in the **GDataY** array. For example:

copy **GMissingDataValue** to **GDataY[3,4]**

or:

copy "*" to **GDataY[3,4]**

The symbol used by **GMissingDataValue** defaults to "*." However, you can copy a different symbol to this variable. The example below defines a dollar sign for the missing value:

copy "\$" to **GMissingDataValue**

16.2.4 Specifying the Number of Data Sets and Data Elements in a Graph

GDataSets and **GDataElements** are required variables for all graphs. **GDataSets** specifies how many data sets are on the graph; **GDataElements** specifies how many data elements per data set are on the graph.

Action: Specifies the number of data sets and elements.

Model

copy TOTAL_DATA_SETS to **GDataSets**

copy TOTAL_DATA_ELEMENTS to **GDataElements**

Where: TOTAL_DATA_SETS is an integer representing the total number of data sets on this graph.

TOTAL_DATA_ELEMENTS is an integer representing the total number of elements per data set on this graph.

If you have more than 10 data sets in any graph in your program, you must edit the BGRAPH.PUB file of the BGraph module and increase the value of the constant **GMaxDataSets**, which has been predefined to 10. If you have more than 20 data elements per data set in any graph in your program, you must edit the BGRAPH.PUB file of the BGraph module and increase the value of the constant **GMaxDataElements**, which has been predefined to 20.

Note: By increasing the values of **GMaxDataSets** and **GMaxDataElements**, the sizes of the internal **BGraph** arrays are enlarged, causing them to take up a greater amount of memory in your program. The normal size of the **BGraph** module is approximately 42,000 bytes. Each additional data set adds approximately 204 bytes; each additional element per data set adds approximately 180 bytes.

16.2.5 Specifying the Decimal Point Character

GDecimalPoint is a string variable that specifies the character to use to represent the decimal point. The default value is ".". Many countries use a different notation. For example, a French application would probably change the value of **GDecimalPoint** to a comma.

Action: Specifies the decimal point character.

Model

copy "," to GDecimalPoint

When the comma is copied to **GDecimalPoint**, the decimal point may be used as the thousands separator.

16.3 Rules and Variables for Certain Graph Types

Some of the graph types have certain requirements that control their usage. They may require that specific variables be set in a certain way, or that the data used for the graph be of a specified type (i.e., for Pie Graphs, the data elements cannot contain both positive and negative numbers). These graph types, their rules, and their variables are discussed in this section.

16.3.1 Overlapping Bar Graph

An overlapping bar graph allows the bars of different data sets to overlap slightly. If only one bar graph is mapped, there is no difference between a regular bar graph and an overlapping bar graph.

GraphOverlap is a boolean variable that specifies whether an overlapping bar graph is to be drawn. It is used only for drawing bar graphs (with the action routine **action GBars**) and is a required variable.

When **GraphOverlap** is set to **false**, no data elements overlap. When **GraphOverlap** is set to **true**, an Overlapping Bar Graph is drawn. **GraphOverlap** is **false** by default. (Illustrations of the various styles of overlapping bar graphs are shown in 16.6, "Illustrations of Graph Types" on page 16-40)

Action: Specifies whether the bar graph is overlapping.

Model

copy { true | false } to GraphOverlap

16.3.2 Three-Dimensional (3-D) Bar Graph

Graph3D is a boolean variable that specifies whether or not a three-dimensional bar graph is to be drawn. It must be used when drawing three-dimensional bar graphs (with the **action GBars**) routine or three-dimensional stacked bar graphs (with the **action GStackedBars** routine).

When **Graph3D** is set to **false**, the bars are drawn in two dimensions: height and width. When **Graph3D** is set to **true**, the bars are drawn in three dimensions: height, width, and depth. **Graph3D** is **false** by default.

Action: Specifies whether the bar graph is three-dimensional.

Model

copy { true | false } to Graph3D

16.3.3 3-D Outline Bar Graph

G3DOutline is a boolean variable that is used for 3D Bar or 3D Stacked Bar graphs. The default value of **true** draws a border on the outer edge of each bar. To reduce drawing time, set the variable to **false**, which eliminates the border. **G3DOutline** is an optional variable.

Action: Specifies whether the 3D bar graph has a border.

Model

copy { true | false } to G3DOutline

16.3.4 Pie Graph

GraphPie is a boolean variable that indicates the validity of drawing a Pie graph. You cannot create a Pie from a data set that has both positive and negative numbers as data elements. If you input both positive and negative data, **GraphPie** returns the value **false**. **GraphPie** is optional.

It is recommended that you check this boolean variable just after the action statement that draws the Pie (**action GPie**), and that you create alternatives for the user if the Pie cannot be drawn.

In the following example, if the data set contains both positive and negative numbers, the Pie is not graphed and the user receives a prompt to select another graph.

```

:
action GPie
if (not (GraphPie)) then    # Pie is not graphed.
  change PromptArea to
    insert "Cannot produce a pie with positive"
          "and negative data.\n"
          "Please select another graph"
  make PromptArea visible
end if
```

GPieDataSet is an integer variable that is used only when creating a Pie graph. It specifies which data set is to be used to draw the Pie. If you do not specify a data set, the default value **1** is used, which will graph the first data set. **GPieDataSet** is an optional variable.

Action: Specifies the pie graph data set to be used.

Model

copy DATA_SET_INDEX to GPieDataSet

Where: DATA_SET_INDEX is an integer representing the data set to be used in drawing the Pie Graph.

Example: copy 3 to GPieDataSet # Use data set 3
 action GPie # to create Pie.

Note: You can also manually label each wedge in a Pie. See 16.4.4.1, "Pie Wedge Labeling" on page 16-20.

16.3.4.1 Width and Height of a Pie Graph Region

For best appearance, the region specified by **GRegion** should not be square. Space is needed to the left and right of the pie for the pie labels. In general, the width of the region should be greater than the height by a factor of 4/3rds. For example:

```
color 26 primary region PieGraph
  size 400 300 at position 100 100
  title bar "Pie Graph"
  system menu
  border
```

If your screen size is **device units** or the parent of the graph region has the **no scale** attribute, use this 4/3rds ratio.

If your screen size is specified as a fixed X and Y value, the ratio of the width and height of the region should be the same as the ratio of the width and height specified in the screen size statement.

If your screen size is **dialog units**, you must calculate the correct ratio at runtime using the following formula:

```
float PieXFactor
:
copy (1.25 * window xsize of desktop * ysize of desktop
      / window ysize of desktop / xsize of desktop)
  to PieXFactor
change PieGraph size to (400 * PieXFactor) 400
```

16.3.5 Stepped Line Graph

GStepLineSet is a boolean variable that must be set to **true** to create a Stepped Line graph. If you do not set this variable to **true**, the Line graph will be drawn with regular lines. **GStepLineSet** is a required variable.

Action: Specifies whether the line graph has stepped lines.

Model

copy { true | false } to GStepLineSet[DATA_SET_INDEX]

Where: DATA_SET_INDEX is an integer representing the data set to be drawn as a stepped line.

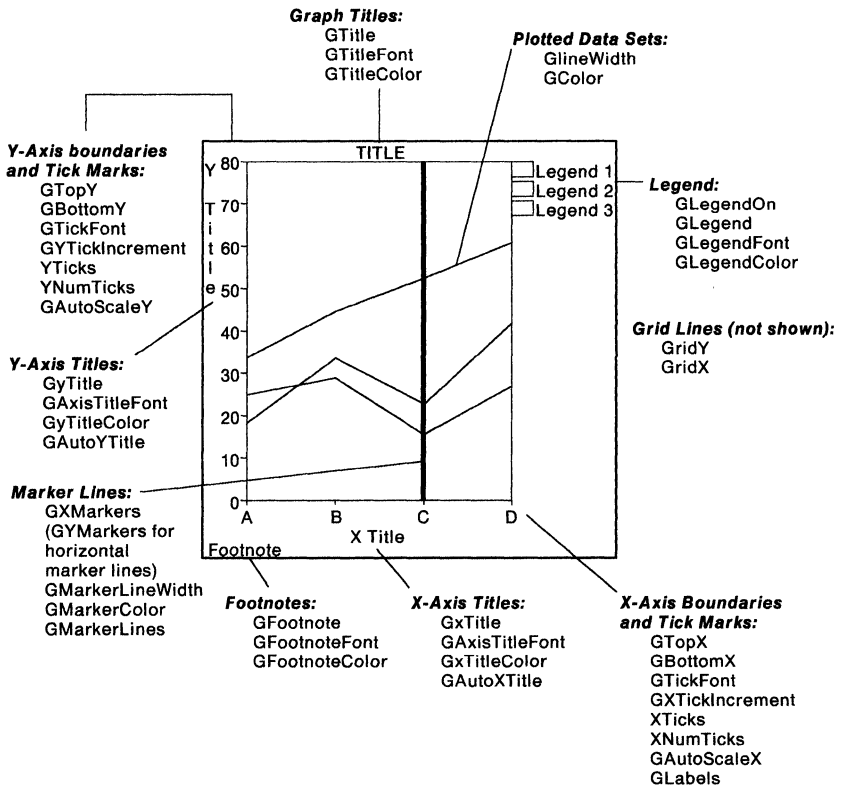
16.3.6 Stacked Line Graph

Stacked Line graphs are limited to 12 data points per data set.

16.4 Customizing a Graph

This section describes the variables that you can specify in your program to control how the graph is displayed. The illustration below shows the names of these variables, all of which are optional.

If you do not specify any titles, footnotes, or legends, the graph fills the space these options would occupy.



16.4.1 Graph Titles

16.4.1.1 Graph Title String

GTitle is a string variable that assigns a title to the entire graph.

Action: Assigns a graph title.

Model

copy TITLE to GTitle

Where: TITLE is a text string representing the title of the graph.

Example: copy "1989 Sales Summary" to GTitle

16.4.1.2 Graph Title Font

GTitleFont is a string variable that assigns a font to the main graph title. The default font is "medium."

Action: Assigns a font to the graph title.

Model

copy FONT_NAME to GTitleFont

Where: FONT_NAME is an EASEL OS/2 EE font name.

Example: copy "ASC79" to GTitleFont

16.4.1.3 Graph Title Color

GTitleColor is a string variable that assigns a color to the main graph title. The default color is 7 (white).

Action: Assigns a color to the graph title.

Model

copy COLOR to GTitleColor

Where: COLOR is an integer or character string representing a color.

Examples: copy 1 to GTitleColor
copy "red" to GTitleColor

16.4.2 Axis Titles

16.4.2.1 Axis Title Strings

GxTitle is a string variable that assigns a title to the X axis. **GyTitle** places the title vertically along the Y axis.

Action: Assigns axis titles.

Model

copy TITLE to { GxTitle | GyTitle }

Where: TITLE is a string representing the title of the axis.

Example: copy "Fiscal Year" to GxTitle

Remarks: If the input data for the Y axis is greater than three digits and automatic scaling is used, EASEL OS/2 EE changes the scale to reflect the scale change in the Y-axis title. When this occurs, EASEL OS/2 EE overrides any string set for the **GyTitle** variable. For example, if the input data is seven digits, the Y-axis title will read "MILLIONS," even if you specified another title for the **GyTitle** variable. The overriding strings are:

THOUSANDS	THOUSANDTHS
MILLIONS	MILLIONTHS
BILLIONS	BILLIONTHS
TRILLIONS	TRILLIONTHS

If your numbers are greater than 100 trillions or smaller than trillionths, EASEL OS/2 EE will *not* override the **GyTitle** variable, but will simply scale the data. In this case, you should set **GyTitle** to reflect the scale change in the Y-axis title, or use manual scaling.

Use **GAutoXTitle** and/or **GAutoYTitle** (below) if you want to use automatic scaling but set your own axis titles.

16.4.2.2 Axis Title Font

GAxisTitleFont is a string variable that assigns a font to the X-axis and Y-axis titles in a graph. The default font is "medium." Note that the font is assigned to *both* axes.

Action: Assigns a font to the axis titles.

Model

copy FONT_NAME to **GAxisTitleFont**

Where: FONT_NAME is a valid EASEL OS/2 EE font name or font reference.

Example: copy "asc1521" to **GAxisTitleFont**

16.4.2.3 Automatic Axis Titles

GAutoXTitle and **GAutoYTitle** are boolean variables that specify whether or not the module should automatically set an axis title. When the variable is set to **true** (which it is by default), the module will replace the appropriate axis title with the title "THOUSANDS," "THOUSANDTHS," etc., when the data is automatically scaled. If you set these variables to **false**, the module will not change the axes titles, even when the data is automatically scaled.

Be careful in disabling the automatic axes titling when the data is automatically scaled. The tick values may not seem to correspond correctly to the data when very high or very low data is being graphed. (Automatic scaling causes the graph to display the data in units such as thousands, thousandths, and so on.)

Action: Disables or enables the axis titles for scaled data.

Model

copy { true | false } to { GAutoXTitle | GAutoYTitle }

Example: copy false to GAutoYTitle

16.4.2.4 Axis Title Colors

GxTitleColor and **GyTitleColor** are string variables that assign a color to the X and Y axis title(s). The default color is **7** (white).

Action: Assigns colors to the axis titles.

Model

copy COLOR to { GxTitleColor | GyTitleColor }

Where: COLOR is an integer or character string representing a color.

Examples: copy 1 to GxTitleColor
copy "red" to GyTitleColor

16.4.3 Axis Boundaries

16.4.3.1 Automatic Axis Boundaries

GAutoScaleY and **GAutoScaleX** are boolean variables that specify whether or not the module should automatically set axis boundaries.

When **GAutoScaleY** or **GAutoScaleX** is set to **true**, EASEL OS/2 EE automatically assigns top and bottom values to the associated axis based on the input data, and will always display a "0" on the Y axis. ("0" will be displayed on the X axis only when X data is used.) **True** is the default for **GAutoScaleY** and **GAutoScaleX**.

Action: Specifies whether to assign axis boundaries.

Model

copy { true | false } to { **GAutoScaleY** | **GAutoScaleX** }

Remarks: To manually scale the graph or disable the display of the "0" on an axis, set **GAutoScaleY** and/or **GAutoScaleX** to **false**. Then, there are two methods by which you can manually scale the graph:

1. Set top and bottom values for the X and/or Y axis (see **GTopY**) and set the tick labels and number of ticks (see **YTicks**).
2. Set top and bottom values for the X and/or Y axis, and set the tick interval (see **GYTickIncrement**). The labels will be filled in by the module relative to the specified increment.

16.4.3.2 Axis Boundary Values

GTopY and **GBottomY** are string variables that assign top and bottom values to the Y axis; **GTopX** and **GBottomX** assign right (top) and left (bottom) values to the X axis. The value of the bottom must be less than that of the top. In order to use these variables, automatic scaling must be disabled. These variables *must* be used when automatic scaling has been disabled.

The module will create a graph with the boundaries specified by **GTopY** and **GBottomY** and/or **GTopX** and **GBottomX**. Data that is too large or too small to fit within the boundaries will not be displayed on the graph. The module will display only that part of the graph that is within the specified boundaries.

Action: Assigns top and bottom values to an axis.

Model

```
copy TOP_VALUE to { GTopY | GTopX }
```

```
copy BOTTOM_VALUE to { GBottomY | GBottomX }
```

Where: TOP_VALUE is an integer representing the highest value on the Y or X axis of the graph.

BOTTOM_VALUE is an integer representing the lowest value on the Y or X axis of the graph.

Example: copy false to GAutoScaleY
copy "50" to GTopY
copy "-10" to GBottomY

16.4.4 Tick and Pie Wedge Labels

16.4.4.1 Pie Wedge Labeling

GPieLabels allows you to disable and enable the labeling of each wedge in a Pie. When **GPieLabels** is **true** (the default), the module labels the Pie wedges with the appropriate percentage. Labeling is disabled when you specify **false** in the following statement.

Action: Disables or enables automatic pie wedge labeling.

Model

```
copy { true | false } to GPieLabels
```

16.4.4.2 Automatic Scaling Tick Mark Labels

GLabels is used to set X-axis tick labels for any graph that is automatically scaled, except for graphs that use **GDataX** (Scatter or Line graphs). When **GDataX** is used, the X-axis tick labels will be generated automatically (based on the data), and **GLabels** will be ignored.

Note: If you use **GLabels** to label Pie wedges with labels other than percentages, **GPieLabels** (above) must be **true**.

Action: Specifies X-axis tick labels for automatically scaled data.

Model

```
copy GROUP_1 to GLabels[ SUBSCRIPT_1 ]
⋮
copy GROUP_n to GLabels[ SUBSCRIPT_n ]
```

Where: GROUP_n is a string representing the label for the nth group of data elements on the X axis or the nth Pie wedge.
SUBSCRIPT_n is an integer representing the subscript for the nth group of data elements on the X axis or the nth Pie wedge.

Example: copy "82" to GLabels[1]
copy "83" to GLabels[2]
copy "84" to GLabels[3]
copy "85" to GLabels[4]

Remarks: If you do not specify any values for **GLabels**, a tick mark is still displayed at the center of each group of data elements, labeled "1", "2", "3", and so on.

In a Pie graph, percentages will be calculated and displayed for each Pie wedge, with call-out lines running from each to its respective percentage. If you did not specify all the **GLabels** array elements, values will be assigned numeric labels.

If you wish to define your own labels to go along with the percentages, set up the pie graph, copy "true" to **GPieLabels**, and then reference action **GPie**. Next, add an enabled key at position 0,0 in the graph region, with a very high priority. Add the labels as text.

16.4.4.3 Tick Mark Label Font

GTickFont is a string variable that assigns a font to the X-axis and Y-axis tick labels. The default font is "medium."

Action: Specifies a font for tick labels.

Model

copy FONT_NAME to GTickFont

Where: FONT_NAME is a valid EASEL OS/2 EE font name or font reference.

Example: copy "large" to GTickFont

16.4.4.4 Number of Tick Marks and Tick Mark Labels

YNumTicks and **XNumTicks** are integer variables representing the number of ticks on the Y and/or X axis. **YTicks** and **XTicks** are string array variables that specify the labels for the ticks. To use these variables, disable automatic scaling and specify new axis boundary values (see 16.4.3, "Axis Boundaries" on page 16-19).

Action: Specifies tick labels for manually scaled data.

Model

copy NUMBER_OF_TICKS to { YNumTicks | XNumTicks }

**copy TICK_VALUE to { YTicks[TICK_NUMBER] |
XTicks[TICK_NUMBER] }**

Where: NUMBER_OF_TICKS is an integer representing the total number of ticks on the X or Y axis. The default number of ticks is 20.

TICK_VALUE is a string value representing the label for the tick mark. TICK_NUMBER is an integer specifying the tick mark. The default range for TICK_NUMBER is 1 through **GMaxDataElements**.

Example: response to ZoomGraphData
 copy false to GAutoScaleY
 copy 105 to GTopY
 copy 100 to GBottomY
 copy 6 to YNumTicks
 copy 100 to YTicks[1]
 copy 101 to YTicks[2]
 copy 102 to YTicks[3]
 copy 103 to YTicks[4]
 copy 104 to YTicks[5]
 copy 105 to YTicks[6]
 action GLines

In the above example, the graph contains six labels, labeled 100 through 105. All graphed values are assumed to be between 100 and 105.

16.4.4.5 Tick Mark Intervals

GXTickIncrement and **GYTickIncrement** are string variables that specify the tick intervals for the Y and/or X axis. The labels are filled in by the module based on the specified increment. To use these variables, disable automatic scaling.

Action: Specifies tick intervals.

Model

copy INCREMENT to { GYTickIncrement | GXTickIncrement }

Where: INCREMENT is an integer or floating point value (such as ".25") specifying the tick increments. You will get unpredictable results if INCREMENT does not contain the same precision of decimal places as in the **GTop** and **GBottom** values for the appropriate axis and if INCREMENT is not evenly divisible into both the top and bottom values specified.

The maximum data point value allowed when **TickIncrement** is used is one billion; the smallest is trillionths.

Example: copy "100" to GYTickIncrement

16.4.5 Legends

16.4.5.1 Legend Display

GLegendOn is a boolean variable that specifies whether or not the legend is to be displayed. When **GLegendOn** is **true**, the legend is displayed. When **GLegend** is **false**, the legend is not displayed, and the graph fills the space otherwise occupied by the legend. **GLegendOn** is **true** by default.

Action: Specifies whether to display the legend.

Model

```
copy { true | false } to GLegendOn
```

Example: copy false to GLegendOn

16.4.5.2 Legend Strings

GLegend is an array of strings representing the names of the data sets being graphed. These names are displayed in the legend.

Action: Specifies data set names in a legend.

Model

```
copy STRING_1 to GLegend[ SUBSCRIPT_1 ]  
⋮  
copy STRING_n to GLegend[ SUBSCRIPT_n ]
```

Where: STRING_n is a string representing the name of the nth group of data sets.

SUBSCRIPT_n is an integer representing the subscript for the nth group of data sets.

Example: copy "Red Product" to GLegend[1]
copy "Green Product" to GLegend[2]
copy "Blue Product" to GLegend[3]

16.4.5.3 Legend Font

GLegendFont is a string variable that assigns a font to the legend. The default font is "medium."

Action: Specifies a font for the legend.

Model

copy FONT_NAME to GLegendFont

Where: FONT_NAME is a valid EASEL OS/2 EE font name or font reference.

Example: copy "large" to GLegendFont

16.4.5.4 Legend Color

GLegendColor is a string variable that assigns a color to the text in the legend. The default color is 7 (white).

Action: Specifies a color for the legend text.

Model

copy COLOR to GLegendColor

Where: COLOR is an integer or character string representing a color.

Examples: copy 1 to GLegendColor
copy "red" to GLegendColor

16.4.6 Footnotes

16.4.6.1 Footnote String

GFootnote is a string variable that assigns a footnote to the entire graph.

Action: Specifies a footnote.

Model

copy FOOTNOTE to GFootnote

Where: FOOTNOTE is a string representing the footnote.

Example: copy "10/86 Release Included" to GFootnote

16.4.6.2 Footnote Font

GFootnoteFont is a string variable that assigns a font to the footnote on a graph. The default font is "medium."

Action: Specifies a Font for the Footnote

Model

copy FONT_NAME to GFootnoteFont

Where: FONT_NAME is a valid EASEL OS/2 EE font name or font reference.

Example: copy "asc59" to GFootnoteFont

16.4.6.3 Footnote Color

GFootnoteColor is a string variable that assigns a color to the footnote in a graph. The default color is 7 (white).

Action: Specifies a color for the footnote.

Model

copy COLOR to GFootnoteColor

Where: COLOR is an integer or character string representing a color.

Examples: copy "red" to GFootnoteColor
copy 1 to GFootnoteColor

16.4.7 Colors

16.4.7.1 Data Set Colors

GColor specifies the color for each data set in any graph except a Pie, where **GColor** specifies the color for each wedge in the Pie. If you do not specify colors, the following default values are used:

Data Set (Data Element for Pie)	EASEL OS/2 EE Color Value	Color
[1]	1	Red
[2]	2	Green
[3]	3	Yellow
[4]	4	Blue
[5]	5	Magenta
[6]	6	Cyan
[7]	7	White
[8]	8	Black
[9]	1	Red
[10]	2	Green
... cycles through colors in same order, to:		
[20]	4	Blue

To ensure that the data is distinguished from the background of the graph, make sure that the background color of the graph region is different from the specified colors in **GColor**.

Action: Specifies colors of plotted data sets.

Model

```
copy COLOR_NUMBER to GColor[ SUBSCRIPT_1 ]  
:  
copy COLOR_NUMBER to GColor[ SUBSCRIPT_n ]
```

Where: COLOR_NUMBER is an integer representing the EASEL OS/2 EE color value of the data set.

SUBSCRIPT_n is an integer representing the subscript for the nth data set.

Example: copy 1 to GColor[1] # Data set 1 is red
copy 2 to GColor[2] # Data set 2 is green
copy 4 to GColor[3] # Data set 3 is blue

16.4.7.2 Red Negative Bars Display

GRedNegativeBars is a boolean variable used only for Bar Graphs to specify whether or not negative values are graphed in red.

Action: Specifies whether or not to graph negative values in red.

Model

```
copy { true | false } to GRedNegativeBars
```

If **GRedNegativeBars** is **true**, all negative values are graphed in red when a Bar graph is displayed. By default, **GRedNegativeBars** is set to **false**, and negative values are graphed in the same color as the positive values in each data set.

16.4.8 Coloring Text in a Graph (Reference)

You can specify the color for text in the following areas of a graph.

To change the color of ...	See ...
Graph title	GTitleColor
X-axis title	GxTitleColor
Y-axis title	GyTitleColor
Legend	GLegendColor
Footnote	GFootnoteColor
Marker lines	GMarkerColor

Tick labels are always drawn in the foreground color of **GRegion**.

16.4.9 Grid Lines

GridX and **GridY** are boolean variables that, when set to **true**, draw grid lines on the X and Y axis. The grid lines are produced by extending the tick marks on the specified axis.

Action: Specifies whether or not to draw grid lines.

Model

copy { **true** | **false** } to { **GridX** | **GridY** }

Remarks: If both **GridX** and **GridY** are set to **true**, the ticks form a cross-grid over the entire graph. **GridX** or **GridY** are **false** by default.

16.4.10 Marker Lines

You can place marker lines anywhere on either axis on a graph. These marker lines highlight a particular set of data points or visually partition points on the graph.

To set the marker lines:

1. Set **GMarkerLines** to **true**.
2. Copy the marker values into the next available data set in **GDataY** and/or **GDataX** (**GDataElements** + 1). For example, if there are three data sets, use data set 4 to store the marker information. The total number of markers allowed on a graph is equal to **GDataElements**.

3. Set the corresponding elements in **GYMarkers** or **GXMarkers** to **true**.

The following is an example of how to set marker lines in your program. This example assumes that **GDataY** contains values in data sets 1, 2, and 3 only. A marker is placed at the plotted data points that are defined by Y values of 75 and 100.

```
copy "75" to GDataY[4,1]      # Copies marker values
copy "100" to GDataY[4,2]     # into next highest
                              # data set.
copy true to GYMarkers[1]     # Enables marker for
                              # element 1.
copy true to GYMarkers[2]     # Enables marker for
                              # element 2.
copy true to GMarkerLines     # Enables drawing.
```

Each of the marker variables is described in detail below.

16.4.10.1 Marker Lines Display

GMarkerLines is a boolean variable that specifies whether or not the graph is to display marker lines. It must be used in conjunction with other marker variables, as shown 16.4.10, "Marker Lines" on page 16-29.

Action: Specifies whether or not to display marker graph lines.

Model

copy { true | false } to **GMarkerLines**

Remarks: When **GMarkerLines** is set to **false** (which it is by default), no marker lines will be drawn on the graph.

16.4.10.2 Data Set Marker Lines Display

GYMarkers and **GXMarkers** are boolean variables that enable or disable markers for the data elements of the marker's data set. The marker's data set is the first **GDataY** or **GDataX** data set that is not used for graphed data.

Action: Specifies whether or not to display marker lines for specific data elements.

Model

```
copy { true | false } to { GYMarkers[ MARKER_ELEMENT_NO ] |  
                           GXMarkers[ MARKER_ELEMENT_NO ] }
```

Where: MARKER_ELEMENT_NO is an integer value representing the subscript for the array.

Examples: # if GDataY[4,3] contains the Y coordinate for a marker
line:

```
copy ".01" to GDataY[4,3]
```

```
copy true to GYMarkers[3]
```

```
# if GDataY[5,1] and GDataY[5,4] each contain Y  
# coordinates for a marker line, then:
```

```
copy true to GYMarkers[1]
```

```
copy true to GYMarkers[4]
```

16.4.10.3 Marker Line Color

GMarkerColor is a string variable that specifies the color for the marker lines displayed on the graph. You can specify only one color for all the marker lines. The default color is 1 (red).

Action: Specifies marker line color.

Model

```
copy COLOR to GMarkerColor
```

Where: COLOR is an integer or character string representing an EASEL OS/2 EE color.

Examples: copy 3 to GMarkerColor
copy "blue" to GMarkerColor

16.4.10.4 Marker Line Width

GMarkerLineWidth is an integer variable that specifies the width of the marker lines displayed on the graph. You can specify only one width for all the marker lines on a graph. The default width is 4 coordinate positions.

The values are device-dependent and are therefore relative. The lowest value that can be specified is 2. The normal range is from 2 to 20; typical values are 4 and 8.

Action: Specifies marker line width.

Model

copy WIDTH to GMarkerLineWidth

Where: WIDTH is an integer value representing the width of the marker line in coordinate positions.

Example: copy 5 to GMarkerLineWidth

16.4.11 Line Widths

GLineWidth is an integer variable that specifies the width of lines plotted in a Line graph. You can specify only one width for all the plotted lines.

The values are device-dependent and are therefore relative. The **GLineWidth** default value is 1.

Action: Specifies line width for line graphs.

Model

copy WIDTH to GLineWidth

Where: WIDTH is an integer value representing the width, in coordinate positions, of the plotted line.

Example: copy 5 to GLineWidth

Remarks: Note that when you increase the line width on a Line graph that contains many elements per data set, the drawing time can double.

16.4.12 Symbols

GPatterns is a string array variable that specifies the type of symbols to be used for each data set plotted on a Scatter graph or a Line graph with Symbols. You can add your own EASEL OS/2 EE patterns to this array.

If you are adding your own EASEL OS/2 EE patterns as symbols, use a template of 16x16 pixels for each symbol (EASEL OS/2 EE pattern) you create, to ensure accurate scaling. Copy the pattern name to the array as shown in the following model.

Action: Specifies patterns for graph symbols.

Model

copy PATTERN_NAME to GPatterns[DATASET_NUMBER]

Where: PATTERN_NAME is a string value representing the name of the EASEL OS/2 EE pattern to be copied to the array.
DATASET_NUMBER is an integer representing the data set for which the pattern is to be displayed.

Example: copy Logo to GPatterns[4]

Remarks: The default values for symbols used are as follows:

Data Set	Symbol	Data Set	Symbol
1	△	6	●
2	□	7	■
3	▲	8	×
4	+	9	□
5	○	10	△

You can also change the order of the default patterns, or add the names of your own patterns (see listing in 16.8, “Public File of the BGRAPH Module” on page 16-50).

16.4.13 Superimposing Two Graphs

GraphOnLast is a boolean variable that determines whether or not to draw a new graph on the axes of the previously drawn graph.

When **GraphOnLast** is set to **false**, each graph is drawn with a new set of axes. When **GraphOnLast** is set to **true**, the two graphs are superimposed and the new graph is plotted on the previous graph's axes. **GraphOnLast** is **false**, by default.

Note: New graph displays only the data that is in the same range as the previously drawn graph. If the values of the new graph are out of range of the values of the previous graph, the scaling is *not* adjusted for the new graph.

Action: Specifies whether to superimpose graphs.

Model

copy { true | false } to GraphOnLast

Example: copy true to GraphOnLast

16.4.14 GMovables: Moving Text in a Graph

GMovables is a class of objects in the BGraph module that includes the legend, axis labels, title, and footnote. By using **GMovables**, you can change the position and ancestry of these objects during runtime.

Model (partial code)

```
string variable CURRENT_GRAPH_PART

class SENSE_REGIONS

response to GMoveables
  copy object in GRegion to CURRENT_GRAPH_PART
  delete SENSE_REGIONS from class SENSE_REGIONS
  add
    sense region SENSE_AREA size (xsize of GRegion) (ysize of GRegion)
    at 0 0 in GRegion priority is 12
  add SENSE_AREA to class SENSE_REGIONS
  begin
    response to SENSE_REGIONS
      change CURRENT_GRAPH_PART to xcoord ycoord
      delete SENSE_AREA
      leave block
  end
```

Where: CURRENT_GRAPH_PART is a string variable that contains the name of the selected object in the graph.

SENSE_AREA is the name of a sense region, defined exactly in the same size and position as the region **GRegion**, described in 16.2.1, "Required Procedures" on page 16-3.

SENSE_REGIONS is the name of a class that will contain SENSE_AREA. It must be present so that a response can be defined.

As the model illustrates, you must do the following in order to move a graph object:

1. Define the string variable CURRENT_GRAPH_PART, which will contain the name of the graph object when it is selected.
2. Define the response to **GMoveables** (as shown in model).
3. Define the "copy object in GRegion to CURRENT_GRAPH_PART" statement, which saves the name of the selected object.
4. At this point, you may want to insert an action statement that tells the user to select a new location for the object selected. For example:

```
change PromptArea to
insert "Select new position."
```
5. Add SENSE_AREA to **GRegion**, so that the user can select the new position.

6. Include the block exactly as shown in the model. This block is activated after the new position has been selected. It moves the selected object to the new position, and then deletes SENSE_AREA.

Notes:

1. If you do not want the user to be able to select a graph object for a specific period of time, disable **GMovables** and enable it later in the program.
2. SENSE_AREA must be added as a child of **GRegion**, so that if **GRegion** is moved or resized, SENSE_AREA is similarly affected.

16.5 Sample Program Using the BGRAPH Module

The following EASEL OS/2 EE program is an example of how to use the Business Graphics module to create a 3-D Bar graph.

screen size 640 480

```
module BGraph                                # Invokes BGraph module.

integer variable Set                          # Holds current data set
                                              # while loading GDataY.

string variable CurrentGraphPart             # Holds selected object.

invisible textual region DataArea            # Place where data will
  window size 32 columns 3 lines             # be extracted from and
  at position 0 0                            # placed in GDataY.
  insert "HARDWARE      24      29      15    27\n:"
  insert "SOFTWARE      34      45      53    61\n:"
  insert "MAINTENANCE   18      33      22    42"

invisible blue region GraphArea              # Region that will
  size 400 250                              # contain the finished
  at position 120 100                       # graph.
  white border

enabled blue region MakeGraph               # A key to create
  size 150 50                              # the graph.
  at position 40 25
  white foreground
  white border
  move to 75 25
  font "large" text center "GRAPH"

enabled red region Quit                    # A key to stop
  size 150 50                              # the program.
  at position 450 25
  white foreground
  white border
  move to 75 25
  font "large" text center "STOP"

invisible enabled sense region SelectArea   # Region to hold moved
  size 400 250                             # GMoveables parts of
  at position 120 100                       # the graph.
  priority is 12
```

```

# The following action routine reads the table of numbers from
# DataArea and loads it into the array GDataY. Once GDataY
# contains the data, the module uses it to create the graph.

action ExtractGraphData is
  copy 1 to Set
  while (Set <= bottom of DataArea) loop
    extract from textual line Set from DataArea
    take word GLegend[Set]                # Title of data set (legend)
    take word GDataY[Set,1]
    take word GDataY[Set,2]                # There are 4 data elements
    take word GDataY[Set,3]                # in this example.
    take word GDataY[Set,4]
    copy (Set+1) to Set
  end loop
  copy (Set-1) to GDataSets                # Calculated number of data sets.

response to start
  copy 4 to GDataElements                # Tell the module the number of
                                         # data elements.
  copy "GraphArea" to GRegion            # Tell the module where to
                                         # display the graph.
  copy "BGraph Example" to GTitle        # Give the graph a main title,
  copy "large" to GTitleFont             # and display it in "large"
                                         # font.
  copy "'82" to GLabels[1]               # Label the data elements
  copy "'83" to GLabels[2]               # along the X axis.
  copy "'84" to GLabels[3]
  copy "'85" to GLabels[4]
  copy true to Graph3D                  # The bar graph will be
                                         # 3-dimensional.
  action ExtractGraphData                # Load the data into GDataY.

response to MakeGraph
  action GBars                            # Tell the module to create
  make GraphArea visible                  # a bar graph.

```

```

response to GMovables
  copy object in GraphArea to CurrentGraphPart
  make SelectArea visible
  enable SelectArea
  begin resumable
    response to SelectArea
      change CurrentGraphPart position to xcoord ycoord
      make SelectArea invisible
      disable SelectArea
      leave block
    end
  end
end

```

```

response to Quit
  exit

```

Note: This example program does not define a primary region. Therefore, to compile this example program, the **-fullscreen** command line option must be used.

16.6 Illustrations of Graph Types

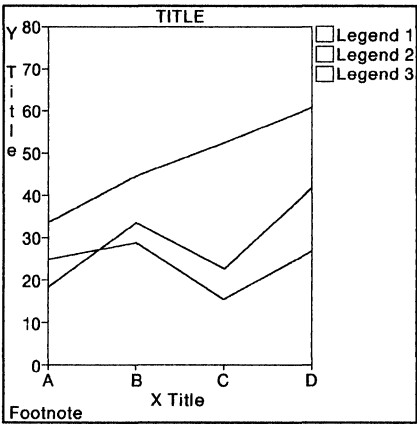


Figure 16-1. Standard Line Graph

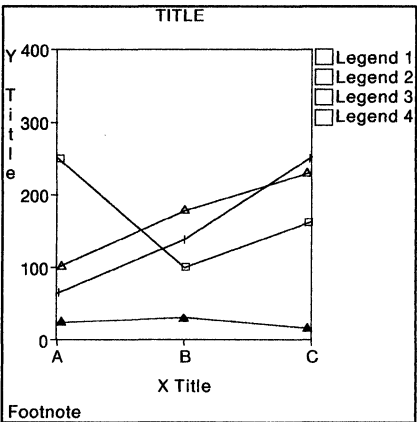


Figure 16-2. Line Graph with Symbols

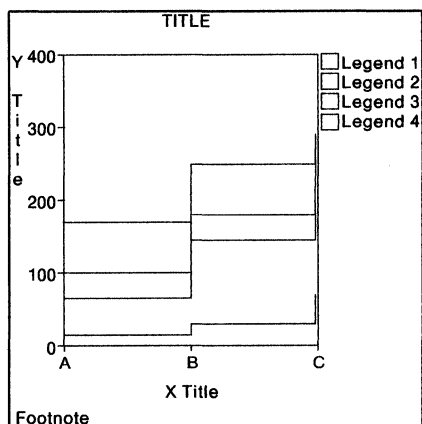


Figure 16-3. Stepped Line Graph

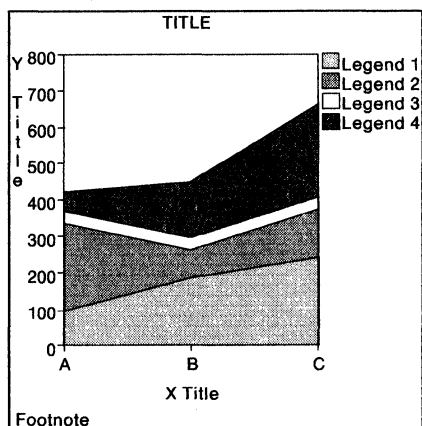


Figure 16-4. Stacked Line Graph

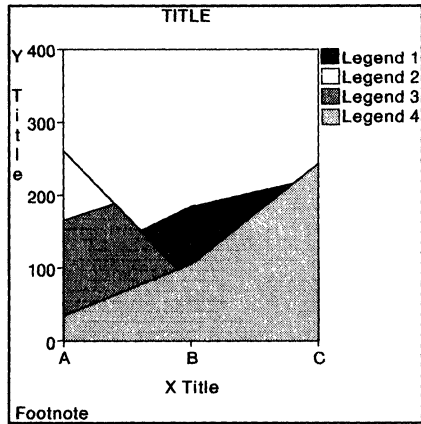


Figure 16-5. Surface Line Graph

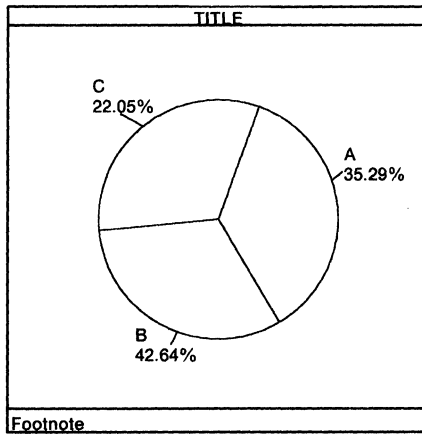


Figure 16-6. Pie Graph

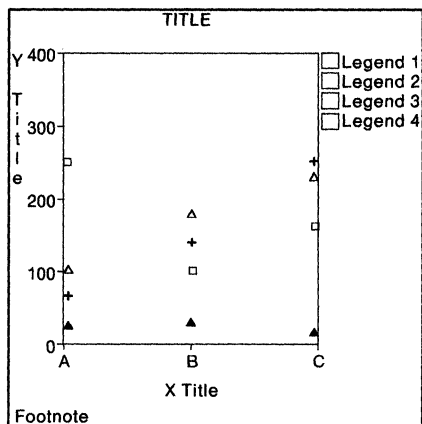


Figure 16-7. Scatter Graph

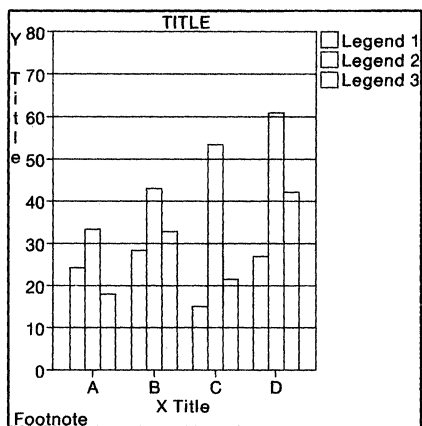


Figure 16-8. Standard Bar Graph

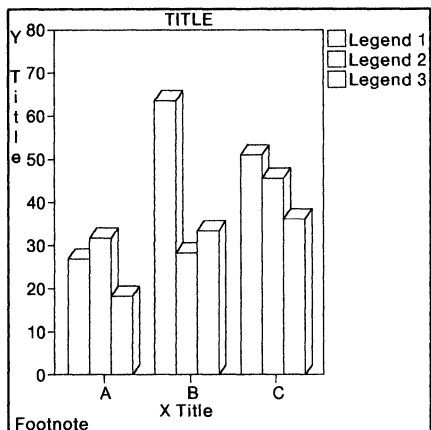


Figure 16-9. 3-D Bar Graph

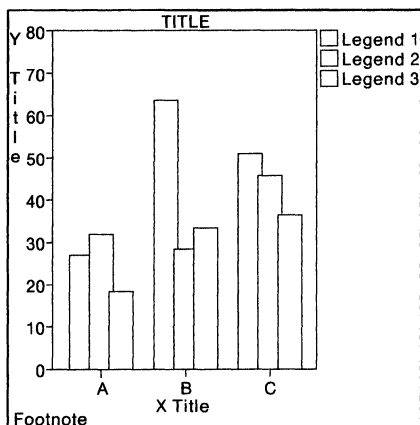


Figure 16-10. Overlapping Bar Graph

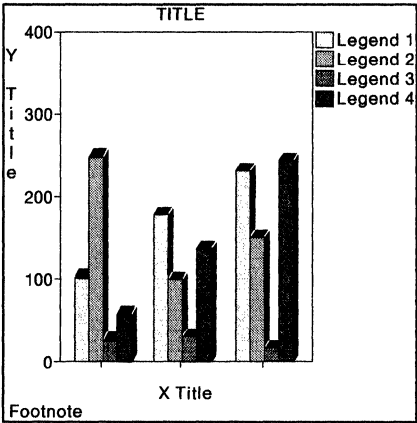


Figure 16-11. 3-D Overlapping Bar Graph

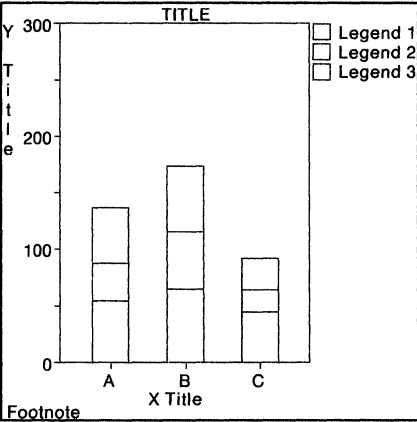


Figure 16-12. Stacked Bar Graph

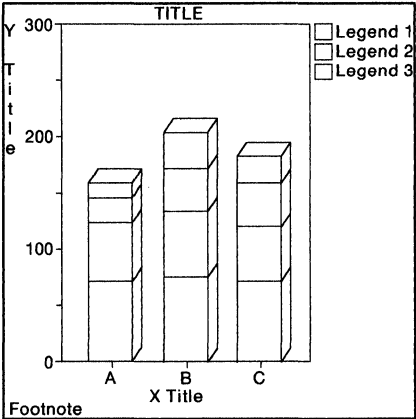


Figure 16-13. 3-D Stacked Bar Graph

16.7 Summary of BGraph Module Variables

Variable Name	Type	Initial Value	Use
GAutoScaleX	boolean	true	Automatic X-axis scaling and labels.
GAutoScaleY	boolean	true	Automatic Y-axis scaling and labels.
GAutoXTitle	boolean	true	Automatic X-axis title.
GAutoYTitle	boolean	true	Automatic Y-axis title.
GAxisTitleFont	string	medium	X-axis and Y-axis title font.
GBottomX	string	" "	X-axis lowest (left) value.
GBottomY	string	" "	Y-axis lowest (bottom) value.
GColor	string array	see 16.4.7.1	Color of plotted data sets or Pie wedges.
GDataElements	integer	20	Data elements per data set.
GDataSets	integer	10	Data sets per graph.
GDataX	string array	" "	X-axis data elements.
GDataY	string array		Y-axis data elements.
GFootnote	string	" "	Footnote.
GFootnoteColor	string	7	Footnote text color.
GFootnoteFont	string	medium	Footnote font.
GLabels	string array	" "	X-axis tick labels.
GLegend	string array	" "	Data set names in Legend.
GLegendColor	string	7	Legend text color.
GLegendFont	string	medium	Legend font.
GLegendOn	boolean	true	Display legend.
GLineWidth	integer	1	Width of lines in Line graph.
GMarkerColor	string	1	Marker lines color.

Variable Name	Type	Initial Value	Use
GMarkerLines	boolean	false	Display marker lines.
GMarkerLineWidth	integer	4	Marker lines width.
GMissingDataValue	string	" * "	Ignore missing data value.
GPatterns	string	see 16.4.12	Patterns for drawn symbols.
GPieDataSet	integer	1	Data set to plot in a Pie graph.
GPieLabels	boolean	true	Automatic Pie wedge labeling.
Graph3D	boolean	false	3-D bars in Bar or Stacked Bar graph.
GraphOnLast	boolean	false	Draw graph on same axes as previous graph.
GraphOverlap	boolean	false	Overlaps bars in Bar graph.
GraphPie	boolean	true	Checks for negative and positive data when a Pie graph is drawn.
GRedNegativeBars	boolean	false	Colors negative-value bars red.
GRegion	string	" "	Name of region to hold graph.
GridX	boolean	false	X-axis grid lines.
GridY	boolean	false	Y-axis grid lines.
GStepLineSet	boolean array	false	Draw stepped lines in Line graph.
GTickFont	string	medium	Tick labels font (both axes).
GTitle	string	" "	Graph title.
GTitleColor	string	7	Graph title color.
GTitleFont	string	medium	Graph title font.
GTopX	string	" "	X-axis highest (right) value.
GTopY	string	" "	Y-axis highest (top) value.
GXMarkers	boolean array	false	X-axis marker lines.
GXTickIncrement	string	" "	X-axis tick interval.
GxTitle	string	" "	X-axis title.
GxTitleColor	string	7	X-axis title color.
GYMarkers	boolean array	false	Y-axis marker lines.

Variable Name	Type	Initial Value	Use
GYTickIncrement	string	""	Y-axis tick interval.
GyTitle	string	""	Y-axis title.
GyTitleColor	string	7	Y-axis title color.
G3DOutline	boolean	true	Draw border around 3-D bars.
XNumTicks	integer	9	Number of X-axis ticks.
XTicks	string array	[1 thru GMaxDataElements] X-axis tick labels.	
YNumTicks	integer	9	Number of Y-axis ticks.
YTicks	string array	[1 thru GMaxDataElements] Y-axis tick labels.	

16.8 Public File of the BGRAPH Module

```
#####
public body BGraph is

integer constant GMaxDataSets is 10    # The maximum number of Data Sets is 10
      GMaxDataElements is 20          # and Data Elements is 20

boolean variable GraphPie is true      # This variable is set to false when a
      # pie is attempted with both negative
      # and positive numbers in the same
      # Data Set, and NO PIE IS PRODUCED.

#####
#
#The following variables must be set by the author before
#calling one of the graphing actions to create the graph.
#
#####

# The following data arrays are set by the user with the current X (GDataX)
# and Y (GDataY) values for the graph. If you are not graphing X,Y data,
# GDataX should be left null (all elements set to "").

string GDataY[1 thru GMaxDataSets, 1 thru GMaxDataElements]
string GDataX[1 thru GMaxDataSets, 1 thru GMaxDataElements]

string variable GRegion                # This specifies what region the graph
      # will be put into.

integer GDataSets                      # Tells how many Data Sets there are,
integer GDataElements                  # and how many Data Elements there are
      # in current graph.

#####
#
#The following are optional parameters which can be set by the
#author to change graph appearance, or data analysis.
#
#####

      # Used to set X axis labels
string variable GLabels[1 thru GMaxDataElements]

      # Used to set legend label for
      # data sets
string variable GLegend[1 thru GMaxDataSets]
boolean variable GLegendOn is true     # Used to enable legends in those
      # graphs that support them

string variable GxTitle                # Used to specify titles for X axis
string variable GTitle                 # Graph
string variable GFootnote              # Footnote
string variable GyTitle                # Y axis (if scale is changed from data
      # will contain scale change
      # i.e. thousands)

string variable GTitleFont             # Fonts used for various titles (default
      GTickFont                       # is medium for all) Tick labels
```

```

    GFootnoteFont      # Footnote
    GLegendFont        # Legend
    GAxisTitleFont     # X and Y axis labels

string GDecimalPoint is "."      #Included for compatibility with prior
                                #versions of BGraph

boolean GAutoScaleY is true      # Tells whether to use autoscale
boolean GAutoScaleX is true      # or X and Y values specified
string GTopY GTopX GBottomY GBottomX # here. If GAutoScaleX or
                                # GAutoScaleY is false then the Top
                                # and Bottom values must be set.

string GXTickIncrement is "."
string GYTickIncrement is "."
string YTicks[1 thru GMaxDataElements] # Y tick labels
string XTicks[1 thru GMaxDataElements] # X tick labels
integer XNumTicks is 9              # number of x tick marks
integer YNumTicks is 9              # number of y tick marks

boolean GAutoYTitle is true        # Tells whether scale override for
boolean GAutoXTitle is true        # X or Y axis should be used instead
                                # of GxTitle or GyTitle.
                                # (true=override GxTitle/or GyTitle)

boolean GraphOnLast is false       # Tells Graph Module to graph new data
                                # on previous graph.

boolean GridX is false             # These variables tell whether Grid
boolean GridY is false             # ticks should be extended over full
                                # graph on X and/or Y axis.

boolean GraphOverlap is false      # These variables are for bar graphs
boolean Graph3D is false           # only. They specify overlapping and/or
                                # 3D styles of bars.

boolean G3DOutline is true         # Outline 3D bars.

boolean GRedNegativeBars is false  # Specifies negative bars should be
                                # drawn red.
                                # Allows color selection for different
                                # Data Sets (for bar and line graphs),
                                # different Data Elements (for pie).
                                # Can be set with any valid
                                # EASEL OS/2 EE color numbers.

string variable GColor[1 thru GMaxDataElements] is
    1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2,3,4

# The GPatterns array may be loaded with names of patterns to be used
# for the drawing of symbols on graphs

string GPatterns[1 thru GMaxDataSets] is
    "%Triangle", "%SquareP", "%TriangleS", "%Cross",
    "%CircleP", "%SCircleP", "%SSquareP", "%CrossX",
    "%Triangle", "%SquareP"

string variable GFootnoteColor is 7 # Allow setting of footnote color
string variable GxTitleColor is 7  # Allow setting of Xtitle color
string variable GyTitleColor is 7  # Allow setting of Ytitle color
string variable GTitleColor is 7   # Allow setting of Graph Title color
string variable GLegendColor is 7  # Allow setting of Legend word color

string variable GMissingDataValue is "" # For missing line data

```

```

# Here for compatibility with prior
# versions:
integer variable GPriority[1 thru 10] is 1,2,3,4,5,6,7,8,9,10

integer variable GPieDataSet is 1 # Specifies which data set should be
# used for pie representation.

boolean variable GPieLabels is true # Specifies whether labels should be
# put around pie slices
boolean variable GStepLineSet[1 thru GMaxDataSets] # Creates a step line
# instead of line
# graph.

integer variable GLineWidth is 1

boolean GMarkerLines is false # true if there are x or y marker lines
integer GMarkerLineWidth is 4 # width of X and Y marker lines
string GMarkerColor is 1 # Color of X and Y marker lines
boolean GXMarkers[1 thru GMaxDataElements] # x markers located
boolean GYMarkers[1 thru GMaxDataElements] # y markers located
# Classes for Moving parts of graph

class GMovables # Class allows you to move the graph

# title, footnote, X axis title, Y axis
# title, and the whole legend block.

#####
#
#One of these actions must be invoked to create a graph.
#
#####

action GBars is # Used to create bar graph.
action "GGraphBars"
action GStackedBars is # Used to create stack bar graph.
action "GStackedBars"
action GLines is # Used to create line graph.
action "GLines"
action GLinesWithSymbols is # Used to create line graph.
action "GLinesWithSymbols"
action GScatter is # Used to create scatter graph.
action "GScatter"
action GSurfaceLines is # Used to create surface line graph.
action "GSurfaceLines"
action GStackedLines is # Used to create stacked line graph.
action "GStackedLines"
action GPie is # Used to create pie graph.
action "GPie"
action GHiLoClose is # Used to create hi-lo-close graph.
action "GHiLoClose"

end public body BGraph

```

Chapter 17. SQL Support

17.1 Introduction

17.1.1 Requirements

Structured Query Language (SQL) Support for EASEL OS/2 EE is a local application for interpreting and executing dynamic SQL statements. It requires Version 1.2 of OS/2 Extended Edition. You should have a general knowledge of OS/2 Extended Edition Database Manager (DBM), and the corresponding SQL implementation.

DBM must be running before this program is started. Furthermore, some of the direct commands described below will require that the user be logged on to OS/2 Extended Edition User Profile Management Services. When this is required, the PM "logon" dialog box will automatically be displayed.

17.1.2 Access Plans

Before any DBM database can be accessed by SQL commands executed by this application, that database must contain an access plan for this application. An access plan is created by a procedure called binding. The bind procedure can be executed by using the DBM command **sqlbind**, along with the ESLSQL.BND file delivered with EASEL OS/2 EE, or it can be executed directly by this application itself. See the **binddb** command in 17.2.3, "binddb Direct Command" on page 17-6 for more information before using ESLSQL.EXE.

17.1.3 Types of Commands

The SQL Support local application executes four types of commands:

- Direct commands
- Configuration commands
- Inquiry commands
- Dynamic SQL commands

17.1.4 Input and Output

Input is read from standard input, and written to standard output (unless overridden by a configuration command). All input and output is line oriented; input commands are not executed until a newline is received.

17.2 Direct Commands

Direct commands cause the program to execute some DBM kernel service, such as starting or stopping a database. These commands do not contain SQL statements.

The SQL Direct Commands are as follows:

Command	Description
----------------	--------------------

startdb	Starts database.
----------------	------------------

stopdb	Stops currently running database.
---------------	-----------------------------------

binddb	Stops currently running database, runs bind routine, and starts new database.
---------------	---

17.2.1 startdb Direct Command

Action: Starts the database.

Model

startdb DATABASE

Remarks: A database must be started before subsequent dynamic SQL statements can be executed. This command (or the **binddb** command, which automatically starts a database) must be issued successfully before any dynamic SQL statements can be processed.

Example: response to start
start local "SQL" "eslsq1"
send "startdb sample\n" to SQL

17.2.2 stopdb Direct Command

Action: Stops the currently running database.

Model

stopdb

Remarks: This command disconnects the local application from a database. It performs an explicit COMMIT.

Example: response to NewDatabase
send "stopdb\n" to SQL
send "startdb payroll\n" to SQL

17.2.3 binddb Direct Command

Action: Stops any database that is currently running, runs the DBM dynamic bind routine, and starts the new database.

Model

binddb DATABASE

Remarks: The EASEL OS/2 EE local SQL application must be connected once to each database that it will access through a procedure called binding. The bind process creates an DBM access plan for the application, which is required before any SQL statements can be processed successfully. An access plan contains information that the DBM uses to process data requests.

The **binddb** command allows the EASEL OS/2 EE program to direct the local application to bind to the database at runtime. This is most useful when the EASEL OS/2 EE application does not know the target database name at compile time. Since running the bind procedure requires special privileges, however, binding at runtime may be inappropriate. (Issuing a **binddb** command requires SYSADM or DBADM authority, and BIND privilege.) If this is the case, the **sqlbind** command provided with DBM can be used. (This command, which also requires privileges, can be issued by the database administrator.) The **binddb** command requires a bind file called ESLSQL.BND, which is supplied with EASEL OS/2 EE.

If an attempt is made to execute SQL statements from this application on a database to which this application is not bound, a -818 error code ("a timestamp conflict occurred") will result.

See the DBM documentation for more information on access plan maintenance and bind procedures.

Example: response to start
start local "SQL" "eslsql"
send "bindfile c:\\eslsql\\n" to sql
send "binddb sample\\n" to SQL

See Also: **bindfile** Configuration Command
Error Handling

17.3 Configuration Commands

Configuration commands allow you to customize the behavior of, or output from, subsequent commands to this program. They are local to the application.

The SQL Configuration Commands are as follows:

Command	Description
prompt	Sets prompt string.
error	Sets error string.
null	Sets null indicator string.
info	Sets information string.
bindfile	Sets location of bind file.
output	Sends subsequent query output to the specified file.
fs	Sets field separator for subsequent queries.
rs	Sets record separator for subsequent queries.
set	Sets display width for the specified column of a subsequent select statement.
maxwidth	Sets a maximum width that will be returned by subsequent widths commands.
startq	Clears all existing column formats (widths), and resets them to their default widths.
verbose	Turns verbose error reporting on or off.
exit	Terminates the SQL support local application.

17.3.1 prompt Configuration Command

Action: Sets the prompt string.

Model

prompt STRING

Remarks: The prompt string is displayed after each command is executed, signaling the completion of that command. When this command is issued, the string is the rest of the command line, including white space, up to a newline. This means the prompt string can be more than one word. To enter leading spaces or tabs, use a leading backslash (\) character. The program always issues a newline immediately following display of the prompt.

This program will not display the prompt until at least one command has been issued. An EASEL OS/2 EE program can set the prompt to a specific string that can be used in a command like the following:

response to line col 1 "SQL>" from SQL

This allows the EASEL OS/2 EE program to set the prompt to a unique string that will not be confused with data resulting from a DBM query. The program can therefore invoke a different response to the prompt than that invoked upon receipt of query results.

Default: "SQL> "

Example: response to start
start local "SQL" "eslsql"
send "prompt Prompt> \n" to SQL

response to line "Prompt> " from SQL
get another command

17.3.2 error Configuration Command

Action: Sets the error string.

Model

error STRING

Remarks: The error string appears at the beginning of each line containing error information. Receipt of such a line signals that a command has not been processed successfully. Several such lines may be issued for a given error (before the prompt is redisplayed); each line will contain the error string.

When this command is issued, the string is the rest of the command line, including white space, up to a newline. This means the error string can be more than one word. To enter leading spaces or tabs, use a leading backslash (\) character.

Default: "Error: "

Example: response to start
start local "SQL" "eslsql"
send "prompt Prompt> \n" to SQL
send "error ERROR! \n" to SQL

response to line "ERROR! " from SQL
add to ErrorWindow
insert at cursor input
make ErrorWindow visible

See Also: **verbose** Configuration Command

17.3.3 null Configuration Command

Action: Sets the null indicator string.

Model

null [STRING]

Remarks: If a selected row contains a column that has an SQL null indicator, and the null indicator is set for that row, this string is displayed in place of the data. This means that the database contains no data for that column in that row. If the display width for that column is set (with the **set** command, described below), and the display width is shorter than the null string width, the column display width takes precedence. In other words, the null string is truncated to match the column width, just as if it were data.

If the null command is issued with no parameter, the null string is set to be empty (none). In this case, when data is retrieved with the null indicator set, the field is empty. This should be used only with a non-null field separator.

When this command is issued, the string is the rest of the command line, including white space, up to a newline. This means the null string can be more than one word. To enter leading spaces or tabs, use a leading backslash (\) character.

Default: "Null"

Example: response to start
start local "SQL" "eslsql"
send "prompt Prompt> \n" to SQL
send "error ERROR! \n" to SQL
send "null NULL!\n" to SQL

response to line "NULL!" from SQL
add to ErrorWindow
insert at cursor "Warning: Null column retrieved"
make ErrorWindow visible

17.3.4 Info Configuration Command

Action: Sets the information string.

Model

info STRING

Remarks: The information string is prefixed to all informational messages (such as those generated by the **widths**, **names**, and **types** commands written out by the SQL application.

When this command is issued, the string is the rest of the command line, including white space, up to a newline. This means the info string can be more than one word. To enter leading spaces or tabs, use a leading backslash (\) character.

Default: "INFO: "

Example: response to start
start local "SQL" "eslsql"
send "prompt Prompt> \n" to SQL
send "info INFO! \n" to SQL

response to line "INFO! " from SQL
add to InfoWindow
insert at cursor input
make InfoWindow visible

17.3.5 bindfile Configuration Command

Action: Sets the location of the bind file.

Model

bindfile PATHNAME

Remarks: See the *Database Manager Programming Guide and Reference* for a description of the binding process.

PATHNAME must exist, and must contain the file "ESLSQL.BND" (which is delivered with SQL Support for EASEL OS/2 EE), or subsequent **binddb** commands will fail. The pathname should *not* include the filename (the program appends the string "\ESLSQL.BND" to the supplied path, and uses that as the bind filename).

The bind file is not opened until a subsequent **binddb** command is issued.

Default: current directory (".")

Example: response to start
start local "SQL" "eslsql"
send "bindfile c:\\eslsql\\n" to SQL
send "binddb sample\\n" to SQL

See Also: **binddb** Direct Command

17.3.6 output Configuration Command

Action: Sends all subsequent query output to the specified file.

Model

output FILENAME { **append** | **destructive** }

Remarks: The **append** and **destructive** keywords have the same behavior as in the text region **write** statement.

To send query output to standard output after having redirected it to a file, use one of the following file names with either **append** or **destructive**: **con**, **con:**, or **stdout**.

Sending query output to standard output will cause it to be sent to the EASEL OS/2 EE program, allowing the EASEL OS/2 EE program to respond to the query output as input for the SQL support local application.

The specified file is not opened or created until a select statement is issued. The file is closed immediately after the last fetch is executed. Therefore, the file can be used as soon as the prompt is returned, after a select statement is issued. If several queries are directed to the same file, use **append**.

Example: response to BeginQuery
 send "output data.out append\n" to SQL
 send "select * from staff\n" to SQL

response to ShowQuery
 send "output stdout append\n" to SQL
 send "select * from staff\n" to SQL
 :

17.3.7 fs Configuration Command

Action: Sets the field separator for subsequent queries.

Model

fs [STRING]

Remarks: The STRING argument is optional. If this command is issued with no STRING argument, the field separator is set to a null string.

When this command is issued, the string is the rest of the command line, including white space, up to a newline. This means the **fs** string can be more than one word. To enter leading spaces or tabs, use a leading backslash (\) character.

Default: ""

Example: response to Query
send "fs |\n" to SQL
send "select * from org" to SQL

response to line from SQL
extract from input
take to "|" FirstColumn

17.3.8 rs Configuration Command

Action: Sets the record separator for subsequent queries.

Model

rs [CHAR]

Remarks: CHAR can be any single character, or one of:

- \t** tab
- \n** newline
- ** backslash

The CHAR argument is optional. If this command is issued with no CHAR argument, the record separator is set to none (no record separator will be output).

Default: \n (newline)

Example: send "rs \\n\\n" to SQL

17.3.9 set Configuration Command

Action: Sets the display width for the specified column of a subsequent select statement.

Model

set COLUMN_NO WIDTH

Remarks: COLUMN_NO is a column number, and WIDTH is an integer. If the width is smaller than the size of the retrieved data, the data is truncated. If it is longer, the data is padded with blanks. If the width is negative (immediately preceded by a minus sign), the data is left justified within the specified width. Otherwise, the column is right justified.

To reset a column width to the default for that column (meaning no padding and no truncation), use a width of 0. To reset *all* columns to their default widths, use the **startq** command.

The first column in a select statement is column 1. The column number must be less than or equal to 255, which is the largest number of columns that can appear in a select statement. The absolute value of WIDTH must be greater than or equal to 0, and less than or equal to the maximum length of an SQL LONG VARCHAR data type (which is 32700).

Example: response to StaffQuery
send "set 1 20 set 2 4 set 3 20\n" to SQL
send "select name, dept from org\n" to SQL

response to line from SQL
extract from input
take to 20 StaffName

See Also: **widths** Inquiry Command

17.3.10 maxwidth Configuration Command

Action: Sets a maximum width that will be returned by subsequent **widths** commands.

Model

maxwidth [WIDTH]

Remarks: WIDTH is an integer greater than 0 and less than or equal to 32767. The WIDTH argument is optional. If this command is issued with no WIDTH argument, the default value of 256 is restored.

Default: 256

Example: response to BigQuery
send "maxwidth 30\n" to SQL
send "widths select * from org\n" to SQL

See Also: **widths** Inquiry Command

17.3.11 startq Configuration Command

Action: Clears all existing column formats (widths), and resets them to their default widths (no padding and no truncation).

Model

startq

Remarks: This command is shorthand for:
set 1 0 set 2 0 ... set 255 0

Example: response to NewQuery
send "startq\n" to SQL

17.3.12 verbose Configuration Command

Action: Turns verbose error reporting on or off.

Model

verbose [on | off]

Remarks: Verbose error reporting displays more information about the nature of an SQL error than the terse two-line message usually printed. This command is useful when testing query syntax.

Default: off

Example: send "verbose on\n" to SQL # used for testing queries

See Also: **error** Configuration Command

17.3.13 exit Configuration Command

Action: Terminates the SQL support local application.

Model

exit

Remarks: This command frees all resources associated with the local application. Send the command before the **stop** command.

Example: send "exit\n" to SQL
stop SQL
exit

17.4 Inquiry Commands

Inquiry commands return informational messages, usually about an SQL select statement (such as the SQL data types, or names, of columns that appear in the statement). Most inquiry commands accept an SQL statement as a parameter.

The SQL Inquiry Commands are as follows:

Command	Description
---------	-------------

display	Displays current configuration state.
----------------	---------------------------------------

names	Causes program to parse the select statement and deliver an informational message describing the name of each unqualified column that will appear in the result table.
--------------	--

types	Causes program to parse the select statement and deliver an informational message describing the SQL data type of each column that will appear in the result table.
--------------	---

widths	Causes program to parse the select statement and deliver an informational message describing the maximum possible length of each column selected, with each column's width expressed as a set command.
---------------	---

17.4.1 display Inquiry Command

Action: Displays the current configuration state (error string, prompt string, etc.) to standard output.

Model

display

Remarks: Each line is prefixed with the information string.

Example: response to Status
send "display\n" to SQL

response to line "INFO!" from SQL
add to InfoWindow
insert at cursor input

See Also: **Info** Configuration Command

17.4.2 names Inquiry Command

Action: Causes the program to parse the select statement and deliver an informational message describing the name of each unqualified column that will appear in the result table.

Model

names SELECT_STATEMENT

Remarks: Qualified columns are described by their position within the select statement. The message has the form:

INFO_STRING NAME1 NAME2 ... NAME_n

This is useful when all columns in a table are selected using the "*" command in a select statement.

Unqualified columns are those columns which have names that are ambiguous without specifying its associated table name, view name, or correlation name. For more information, see the *IBM OS/2 Extended Edition Database Manager SQL Reference* publication.

Example:

```
response to Names
  send "names " QueryString "\n" to SQL
  begin guarded
    response to line from SQL
    add to ReportWindow
    insert at cursor input
    leave block
  end
```

17.4.3 types Inquiry Command

Action: Causes the program to parse the select statement and deliver an informational message describing the SQL data type of each column that will appear in the result table.

Model

types SELECT_STATEMENT

Remarks: The message has the form:
INFO_STRING TYPE1 TYPE2 ... TYPEn
The type strings delivered are as follows:

Type String	Description
date	date
time	time
timestamp	timestamp
varchar	varying length character
longvar	long varying length character
char	fixed length character
float	floating point
decimal	packed decimal
integer	long integer
smallint	small integer
graphic	fixed length graphics (for DBCS)
vargraph	varying length graphics (for DBCS)
longgraph	long varying length graphics (for DBCS)

Example: response to Types
send "types select * from org\n" to SQL
begin guarded
response to line from SQL
add to TypesWindow
insert at cursor input
leave block
end

17.4.4 widths Inquiry Command

Action: Causes the program to parse the select statement and deliver an informational message describing the maximum possible length of each column selected, with each column's width expressed as a **set** command.

Model

widths SELECT_STATEMENT

Remarks: The message has the form:

```
INFO_STRING set F1 WIDTH set F2 WIDTH ... set Fn WIDTH
```

For each column in the select statement, a **set** command appears on the information message line. The message is terminated with a newline.

The widths values are positive integers for columns with SQL data types float, decimal, integer, and smallint. Otherwise, the widths are negative.

The EASEL OS/2 EE program may respond to this command by stripping off the leading information string, and resubmitting the line as a command, followed by submitting the SELECT_STATEMENT as a command. This has the effect of arranging the output in columns for this query, with all numeric columns right justified and all character columns left justified, since the output width for each column is set to the maximum length of data for that column.

Example:

```
response to OrgQuery
  send "info INFO! \n" to SQL
  copy "select * from org\n" to QueryString
  send "widths " QueryString to SQL
response to line "INFO! " from SQL
  extract from input
    skip by 7                # skip over INFO string
    take to last SetCommand
  send SetCommand "\n" to SQL
```

See Also: **maxwidth** Configuration Command
set Configuration Command

17.5 Dynamic SQL Commands

Any input not recognized as a direct, configuration, or inquiry command is taken to be a dynamic SQL command, and is executed as such. If the SQL command is a select statement, the results of the query are written out according to the current configuration state.

17.6 Error Handling

The EASEL OS/2 EE SQL support local application generates error strings and/or error codes when an error is detected. Some errors are detected by the local application directly, and some are detected by the DBM.

17.6.1 Error Detected by Local Application

If the error is detected by the local application (such errors include direct, configuration, and some inquiry command errors), a single line (beginning with the configured error string and containing a description of the error) is returned. This line is followed by the prompt string. For example, if the **startdb** command is issued with no database name, the following error is issued by the application:

```
ERROR: syntax error in startdb command
SQL>
```

17.6.2 Error Detected by OS/2 Extended Edition Database Manager

If the error is detected by the OS/2 Extended Edition Database Manager (for example, an SQL syntax error), the first line contains a description of the error, and the second line contains the code returned by the OS/2 Extended Edition Database Manager. These lines are followed by the prompt string. For example, an attempt to execute an SQL command on a database that has a timestamp conflict because the EASEL OS/2 EE local application has not been bound to that database will produce the following error:

```
ERROR: SQL PREPARE failure
ERROR: Return Code: -818
SQL>
```

Note: Timestamp conflict errors are not generated when the **startdb** command is issued. An actual database access attempt must be made to trigger this error.

Both lines begin with the configured error string. (In these examples, the default error string "ERROR: " is used). The application issues the prompt after issuing the two error lines. The EASEL OS/2 EE program can use the prompt to detect whether one or two error messages are delivered. (If the **verbose on** configuration command has been issued, more than two lines of error information may be displayed. We recommend this option only for EASEL OS/2 EE application development.)

The "-818" in the example above is the OS/2 Database Manager return code. The meaning of each return code is listed in the *OS/2 Extended Edition Database Programming Guide and Reference*. Since there are many such codes that can be returned, we recommend that certain common

errors be intercepted by the EASEL OS/2 EE program. These errors include the following:

Error Code	Indicates
-818	A timestamp conflict. Most often indicates that the program has not been bound to the target database.
-1032	The OS/2 Extended Edition Database Manager has not been started.

The sample EASEL OS/2 EE code below provides a framework for recognizing these errors. It assumes the default prompt and error strings, and simply displays the error messages in a textual region.

Example:

```
response to line col 1 "ERROR: " from SQL # Respond to
                                           # the first
                                           # error line
add to ErrorWindow insert at cursor input
begin guarded                             # Look for a second error
                                           # line or a prompt
response to line col 1 "ERROR: " from SQL
extract from input take to "Code:" FailMessage
skip by 5
take number FailCode
if (FailCode = -818) then
copy "A timestamp conflict occurred: "
FailCode "\n" to FailMessage
# If appropriate, a "binddb" command can
# be issued here. However, note that issuing
# additional commands to the local
# application will result in the
# application issuing additional prompts
# (or errors messages) that would need to
# be recognized.
```

```

else
    if (FailCode = -1032) then
        copy "No START DATABASE MANAGER COMMAND
            was issued: "
            FailCode "\n" to FailMessage
    else
        copy "Error Number: "
            FailCode "\n" to FailMessage
    end if
end if
add to ErrorWindow insert at cursor FailMessage
begin guarded      # look for the prompt after
                   # the second error line
response to line col 1 "SQL> " from SQL
    leave block
end
leave block

response to line col 1 "SQL> " from SQL
    # after first error line
    leave block
end
make ErrorWindow visible

```

17.7 Command Line Options

You can use the SQL Support local application program as a stand-alone program for testing. At the OS/2 command line, type:

```
eslsql
```

You can include one of the following options:

Option	Meaning
-h	Displays a help/usage message and terminates the program.
-t	Sets the standard output translation mode to "translated" (newlines are converted to cr/lf). The default for output is "untranslated". This command does <i>not</i> affect output to a file, and is most useful for testing from the OS/2 command line.

17.8 Notes on Using SQL Support

17.8.1 Customizing Strings

For the **prompt**, **error**, **null**, **info**, and **fs** commands, a backslash (\) character is ignored if it is the first non-white space (space or tab) character after the keyword. You can use this to enter a string parameter containing leading tabs or spaces. For example, to set the prompt to the following:

```
C: > (two spaces followed by a greater than symbol)
```

use the command

```
prompt \ >
```

To enter a string that begins with a backslash, use two backslashes. For example, to set the prompt to the following:

```
C:\ > (a backslash, two spaces, and a greater than symbol)
```

use the command

```
prompt \\ >
```

17.8.2 Using Output as an Input Command

Be careful when using the output of the format command blindly as an input command.

Columns with SQL data type:	Return as a width:
VARCHAR	"-4000"
LONG VARCHAR	"-32700"

If this output is resubmitted as a command, these columns will be padded to these widths, regardless of the actual amount of data the column contains, resulting in large quantities of output.

17.8.3 LONG VARCHAR Allocation

Each column of type LONG VARCHAR that is specified in a select statement will require a 32,700 byte segment to be allocated. However, only one such segment will be allocated for each such column, no matter how many rows are fetched.

17.8.4 Increasing Memory

If you expect to receive back data consisting of very large records, you can increase the **-g** size accordingly.

17.8.5 Requirement for Dynamic Commands

In order for the **startdb**, **stopdb**, and all dynamic SQL commands to succeed, the DBM must be running. The only command that will attempt to start DBM itself is the **binddb** command.

17.8.6 Statement Size

The limit on statement size is 32765; this is the largest dynamic SQL statement that can be executed.

17.8.7 Case and White Space

Case and white space are not relevant for direct, inquiry, and configuration commands, except as noted under particular commands.

17.8.8 Shared Mode

Databases are always started in DBM shared mode.

17.8.9 Isolation Level

The SQL Support application uses DBM isolation level "REPEATABLE READ."

17.8.10 Standard Output and Standard Error

Error messages and the **display** command always go to standard output. Only bad-command-line-argument messages go to standard error.

17.8.11 Determining Number of Records

Use the SQL command "select count (*) from" in order to determine how many records will be in the results table for a given query.

17.8.12 Error Message

An error message will be generated if a column that has been marked "for bit data" appears in a select statement, and the query will not be processed.

17.8.13 Graphic Data Types

Be careful when selecting columns with SQL data types GRAPHIC, VARGRAPHIC, and LONG VARGRAPHIC, as these require DBCSs to be displayed. EASEL OS/2 EE will display these as single byte characters.

Chapter 18. Audio Support

18.1 Overview

EASEL OS/2 EE audio support allows you to capture and play back sound stored as audio files on your IBM OS/2 Extended Edition programmable workstation hard disk. Using EASEL OS/2 EE audio support, you can record and play back an audio file, optionally pausing during either process. The programmed requests to play or record can optionally be queued, which for example makes it possible to announce when recording is about to begin with an audible prompt such as, "When you hear the beep, please state your name and address." Your program can also detect when the playing or recording process has either begun or ended and execute a specific response to any of those events. For convenience, EASEL OS/2 EE audio support also provides functions to query total and elapsed audio file playing times, to adjust the volume and speaker balance during playback, and to erase audio files.

EASEL OS/2 EE audio support is provided in the form of subroutines in an EASEL OS/2 EE dynamic link library (DLL) that can be called from within your EASEL OS/2 EE program. The subroutines control the IBM Audio Capture and Playback Adapter/A (ACPA), which must have been installed in your workstation prior to using EASEL OS/2 EE audio support. Additional audio equipment such as a microphone, amplified speakers, or headphones is optional, but would typically be used in conjunction with the ACPA to enhance the volume and quality of the sound.

Note: External speakers are strongly recommended; even at maximum volume level the sound output of the built-in workstation speaker is not strong. If used, external speakers must either be attached to an external amplifier or else have internal amplification.

The ACPA has input and output jacks similar to home stereo equipment. By plugging in a microphone you can record your own voice or ambient sound in the room. You can also capture analog sound from any electronic source that can be fed into the ACPA, for example audio cassette tape, compact discs, records, video discs, video cassette tape, or radio or television tuners. When you play the sound back, it always comes through the speaker in your workstation, but it will also play back through headphones or amplified speakers if they are plugged into the ACPA.

18.1.1 Digitized Audio Files

The ACPA captures and stores sound as digitized audio files. EASEL OS/2 EE audio support creates two files, referred to in this chapter as a *file set*, for each segment of recorded sound: a control file with a file extension of *.lau*, and the digitized sound file with a file extension of *.lad*. These files can be maintained like any other OS/2 Extended Edition file, for example,

they can be copied to diskette or to a LAN server making it easy to share sound files among workstations.

The audio files generated for even short segments of sound can be quite large. If you are planning an audio application, be sure to have ample free disk space available. One second of recorded sound will use from 5.5 to 22KB of audio file space, depending upon the sound quality desired for the recording; therefore each minute of recorded sound can require approximately from .4 to 1.4MB For more detailed information, see the `SOUND_QUALITY` parameter on the **AUDRecord()** subroutine call.

18.1.2 Audio Utility

EASEL OS/2 EE audio support comes with an audio utility, **AUDIO.EBI**, an EASEL OS/2 EE program that performs the major EASEL OS/2 EE audio support functions. The audio utility lets you get started right away using EASEL OS/2 EE audio support; you can use it to record, play, and erase audio files without having to write your own EASEL OS/2 EE program to do so. Verbal instructions are provided with the audio utility. To hear the instructions, invoke the audio utility using the following command:

```
ease\ audio
```

and then select the Help pushbutton.

Note: Remember that the IBM Audio Capture and Playback Adapter/A must be installed in your workstation prior to using EASEL OS/2 EE audio support and the audio utility.

18.1.3 Include File

The EASEL OS/2 EE audio support DLL has a corresponding include file that you must reference your EASEL OS/2 EE program before you can invoke any of the audio subroutines the DLL contains.

Your EASEL OS/2 EE program must contain the following line:

```
include "es\audio.inc"
```

The include file declares the EASEL OS/2 EE audio support subroutines and argument types. It also contains definitions for EASEL OS/2 EE audio support integer constants, which are used both to set variables for some subroutine calls and as filters for audio stimulus event notifications. For detailed information about these constants, see both 18.1.5, "Responding to Audio Events" on page 18-4, and 18.2, "EASEL OS/2 EE Audio Support Subroutines" on page 18-6.

18.1.4 Audio Dynamic Link Library

The EASEL OS/2 EE audio support dynamic link library must be identified in your program as a stimulus DLL—one that will send messages to the EASEL OS/2 EE Runtime Facility.

Your EASEL OS/2 EE program must contain the following line:

```
stimulus AUDIO_EVENT "eslaudio.dll"
```

where AUDIO_EVENT is any valid EASEL OS/2 EE identifier you've chosen to denote messages that represent audio events.

18.1.5 Responding to Audio Events

Your EASEL OS/2 EE program is notified via the **response to stimulus** mechanism when certain audio events occur. You can write a response definition for every audio event, or else write responses for some events and ignore others, with the exception of the audio error event to which your program must respond. A parameter is also sent with every event notification message to give additional information about the event.

There are three different event notifications or stimuli that the EASEL OS/2 EE audio support DLL can generate. You respond to any of these events using the EASEL OS/2 EE **response to stimulus** statement, in the following format:

Model

```
response to stimulus AUDIO_EVENT  
    [AUD_Start [ AUD_Play | AUD_Record ] ]
```

```
response to stimulus AUDIO_EVENT  
    [AUD_Stop [ AUD_DiskFull | AUD_OpenError ] ]
```

```
response to stimulus AUDIO_EVENT  
    [AUD_End [ AUD_Play | AUD_Record ] ]
```

In the above model, AUDIO_EVENT can be any valid EASEL OS/2 EE identifier that you have assigned in the stimulus declaration to denote an incoming message from the EASEL OS/2 EE audio support DLL.

The stimulus events and event parameters listed in the above model are EASEL OS/2 EE integer constants, predefined for use in EASEL OS/2 EE audio support. These constants are optionally used as “event filters” and “parameter filters” on the **response to stimulus** statement to limit a given response definition to a specific audio event. In the following example, a window is made visible when a specific audio event—the completion of playback—occurs:

```
response to stimulus AUDIO_EVENT AUD_End AUD_Play
    make Window1 visible
```

If you code an audio **response to stimulus** statement omitting the stimulus event, as in the following example:

```
response to stimulus AUDIO_EVENT
```

then all audio events will trigger that response statement.

The meanings of the EASEL OS/2 EE audio support integer constants, when used as event filters and parameter filters, are listed below.

AUD_Start The playback or record audio function is being started. This event notification is sent just before the function begins. One of two parameter filters can be used with this event:

AUD_Play Playback is starting.

AUD_Record Record is starting.

AUD_Error An error has occurred during playing or recording. One of two parameter filters can be used with this event:

AUD_DiskFull There was not enough disk space to hold the audio files being created during the recording process.

AUD_OpenError The audio files that you asked to play or record could not be opened.

Your program should not ignore the **AUD_Error** event notification—that is, the program should contain **response to stimulus ... AUD_Error AUD_DiskFull** and **response to stimulus ... AUD_Error AUD_OpenError** statements and appropriate action statements. Calls to the playback and recording sub-routines are queued; actual playback or recording executes asynchronously to its call. Therefore, the **AUD_Error** event notification should be monitored because it is the only means your program has of detecting that an error has occurred during playback or recording.

AUD_End The playback or record audio function has completed. This event notification is sent just after completion. One of two parameter filters can be used with this event:

AUD_Play Playback has just completed.

AUD_Record Recording has just completed.

18.2 EASEL OS/2 EE Audio Support Subroutines

EASEL OS/2 EE audio support consists of 13 subroutines callable by the EASEL OS/2 EE programmer:

Subroutine	Function
AUDClose()	Terminates an EASEL OS/2 EE audio support session.
AUDErase()	Removes an audio file set from disk.
AUDOpen()	Initializes an EASEL OS/2 EE audio support session.
AUDPause()	Pauses during audio file playback or recording.
AUDPlay()	Plays an audio file.
AUDQueryBusy()	Determines whether EASEL OS/2 EE audio support is busy.
AUDQueryElapsedTime()	Determines the elapsed playing or recording time of an audio file.
AUDQueryPlayingTime()	Determines the total playing time of an audio file.
AUDRecord()	Creates an audio file.
AUDResume()	Resumes playing or recording an audio file. (Use after AUDPause() .)
AUDSetBalance()	Adjusts the speaker balance during audio file playback.
AUDSetVolume()	Adjusts the speaker volume during audio file playback.
AUDStop()	Stops the playing or recording of an audio file.

EASEL OS/2 EE audio support subroutines operate the same way as other EASEL OS/2 EE subroutines. To invoke an EASEL OS/2 EE audio support subroutine from your EASEL OS/2 EE program, use the **call** statement. The **call** statement syntax is the same for internal EASEL OS/2 EE subroutines and EASEL OS/2 EE audio support subroutines:

```
call SUB_NAME( ARG1[, ARG2]... )
```

where SUB_NAME is the identifier for a previously defined subroutine, and ARG1[, ARG2] are variables you have defined to contain the subroutine parameters. You must use variables as argument list parameters in an EASEL OS/2 EE audio support subroutine call; literals or constants cannot be used. Also, the type and number of arguments after the subroutine name must match the type and number of arguments in the declaration.

18.2.1 Setting Argument Values

The integer constants listed in the table are predefined for use in EASEL OS/2 EE audio support. They can be used to set values for subroutine calls by copying them into integer variables you have defined to be used as subroutine arguments. Of course, any other variables, constants, or literals can also be used to set subroutine argument values as long as the resulting value is valid for the argument.

18.2.2 Return Codes

If an EASEL OS/2 EE audio support subroutine encounters an error condition, the EASEL OS/2 EE built-in function **errorlevel** is set to the number corresponding to the error found. **errorlevel** will contain a value of zero (0) when the subroutine executes without error.

In your EASEL OS/2 EE program, you should refer to an error by its corresponding integer constant rather than its number; integer constants representing errors are listed under the *Returns* section of each subroutine description.

The EASEL OS/2 EE audio support return code values and integer constants are defined in the EASEL OS/2 EE audio support include file (**eslaudio.inc**).

Constant	Value	Where Used
AUD_Beep	1	AUDRecord()
AUD_BusyPausing	8	AUDQueryBusy()
AUD_BusyPlaying	4	AUDQueryBusy()
AUD_Recording	6	AUDQueryBusy()
AUD_DefaultBalance	50	AUDPlay() AUDSetBalance()
AUD_DefaultQuality	1	AUDRecord()
AUD_DefaultVolume	100	AUDPlay() AUDSetVolume()
AUD_DiskFull	22	As parameter filter in response to stimulus
AUD_End	4	As event filter in response to stimulus
AUD_Error	7	As event filter in response to stimulus
AUD_Interrupt	1	AUDPlay() AUDRecord()

Constant	Value	Where Used
AUD_Line	3	AUDRecord()
AUD_LineLeft	1	AUDRecord()
AUD_LineRight	2	AUDRecord()
AUD_Mike	0	AUDRecord()
AUD_MikeLow	4	AUDRecord()
AUD_MusicQuality	1	AUDRecord()
AUD_NoBeep	0	AUDRecord()
AUD_NoPurge	0	AUDStop()
AUD_NotBusy	0	AUDQueryBusy()
AUD_OpenError	8	As parameter filter in response to stimulus
AUD_Play	1	As parameter filter in response to stimulus
AUD_Purge	1	AUDStop()
AUD_Queue	4	AUDPlay() AUDRecord()
AUD_Record	2	As parameter filter in response to stimulus
AUD_Return	2	AUDPlay() AUDRecord()
AUD_Start	3	As event filter in response to stimulus
AUD_StereoQuality	3	AUDRecord()
AUD_VoiceQuality	2	AUDRecord()

18.2.3 AUDClose()

The **AUDClose()** subroutine is called to terminate an EASEL OS/2 EE audio support session. This subroutine has no arguments.

Note: After **AUDClose()** subroutine has executed successfully, the only audio subroutine valid to call is **AUDOpen()**.

AUDClose() is declared in the EASEL OS/2 EE audio support include file (**esludio.inc**) according to the following syntax:

```
subroutine AUDClose()  
    library "esludio"
```

18.2.3.1 Arguments

None

18.2.3.2 Call

A call to **AUDClose()** is made according to the following syntax:

Model

call AUDClose()

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	AUD_NORMAL_COMPLETION Normal completion.
	4	AUD_ERR_ALREADY_DONE EASEL OS/2 EE audio support has already been closed.

18.2.4 AUDErase()

The **AUDErase()** subroutine is called to remove an audio file set from the disk. One call to this subroutine removes both the audio control file (extension **.lau**) and the digitized sound file (extension **.lad**). This subroutine is provided as a convenient way to delete audio files from within your EASEL OS/2 EE program, however audio files can also be removed from the disk using the regular OS/2 Extended Edition ERASE or DEL commands.

AUDErase() is declared in the EASEL OS/2 EE audio support include file (**eslaudio.inc**) according to the following syntax:

```
subroutine AUDErase( string:AUDIO_FILENAME )  
    library "eslaudio"
```

18.2.4.1 Arguments

AUDIO_FILENAME is a string variable that must contain the name of the audio file set to be erased; do not specify the file extension. This argument cannot be null. EASEL OS/2 EE audio support will automatically delete both files in the audio file set: the control file with an extension of **.lau**, and the digitized sound file with an extension of **.lad**.

18.2.4.2 Call

A call to **AUDErase()** is made according to the following syntax:

Model

```
call AUDErase( AUDIO_FILENAME )
```

Returns:	errorlevel	Error Integer Constant/Explanation
	0	AUD_NORMAL_COMPLETION Normal completion.
	8	AUD_ERR_OPEN Specified file could not be found.
	20	AUD_ERR_INVALID_PARMS Invalid argument list.

18.2.5 AUDOpen()

The **AUDOpen()** subroutine is called to initialize an EASEL OS/2 EE audio support session so that the other audio subroutines can be called.

AUDOpen() also provides the pathname specifying where the audio files are located.

Note: **AUDOpen()** must be the first audio subroutine called.

AUDOpen() is declared in the EASEL OS/2 EE audio support include file (**eslaudio.inc**) according to the following syntax:

```
subroutine AUDOpen( string:AUDIO_PATHNAME )  
    library "eslaudio"
```

18.2.5.1 Arguments

AUDIO_PATHNAME is a string variable that you must supply, indicating where the audio files are located. Setting the argument to a null string value ("") indicates that the value in the OS/2 Extended Edition system environment variable DPATH should be used to locate the audio files.

18.2.5.2 Call

A call to **AUDOpen()** is made according to the following syntax:

Model

```
call AUDOpen( AUDIO_PATHNAME )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	AUD_NORMAL_COMPLETION Normal completion.
	8	AUD_ERR_OPEN EASEL OS/2 EE audio support could not be initialized.

Example:

```
# Pathname for audio files  
string MyAudioPath is "c:\\easel-ee"  
:  
call AUDOpen( MyAudioPath )
```

18.2.6 AUDPause()

The **AUDPause()** subroutine is called to pause during playback or recording of an audio file. This subroutine has no argument. The audio file currently being played or recorded is paused.

AUDPause() is declared in the EASEL OS/2 EE audio support include file (**eslaudio.inc**) according to the following syntax:

```
subroutine AUDPause()  
library "eslaudio"
```

18.2.6.1 Arguments

None

18.2.6.2 Call

A call to **AUDPause()** is made according to the following syntax:

Model

```
call AUDPause()
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	AUD_NORMAL_COMPLETION Normal completion.
	4	AUD_ERR_ALREADY_DONE There is no audio file to pause.

18.2.7 AUDPlay()

The **AUDPlay()** subroutine is called to play an audio file created by using the ACPA (from an EASEL OS/2 EE or non-EASEL OS/2 EE program). The sound plays back through the built-in workstation speaker. It will also play back through an amplifier and/or speakers if they are plugged into the ACPA. External speakers are strongly recommended; even at maximum volume level the sound output of the built-in workstation speaker is not strong. If the workstation speaker is your only speaker, then it is recommended that you set the VOLUME argument to a value of 100 and the BALANCE argument to a value of 0.

Note: If the audio file you ask to play is not found or cannot be opened, an **AUD_Error AUD_OpenError** event notification will be signalled to your program; your program should contain a **response to stimulus** statement monitoring for this event.

AUDPlay() is declared in the EASEL OS/2 EE audio support include file (**esludio.inc**) according to the following syntax:

```
subroutine AUDPlay( string:AUDIO_FILENAME
                  integer:PLAY_START_POSITION,
                  integer:PLAY_STOP_POSITION,
                  integer:VOLUME,
                  integer:BALANCE,
                  integer:BUSY_ACTION )

library "esludio"
```

18.2.7.1 Arguments

AUDIO_FILENAME is a string variable that must contain the name of the audio file to be played; do not specify the file extension. This argument cannot be null. AUDIO_FILENAME actually refers to two files with same name but with different extensions: a control file with an extension of **!au** and a file containing the digitized representation of the sound, with an extension of **!ad**.

PLAY_START_POSITION is an integer value (expressed in milliseconds) indicating the position in the audio file at which playing should begin. A zero value means start at the beginning.

PLAY_STOP_POSITION is an integer value (expressed in milliseconds) indicating the position in the audio file at which playing should end. A zero value means play to the end of the file.

VOLUME is an integer value ranging from 0 to 100 indicating the relative volume at which to play back the file. If this argument is set to **AUD_DefaultVolume**, then a value of 100 is used.

Notes:

1. If the built-in workstation speaker is the only speaker, then for best results set VOLUME to 100 and BALANCE to 0.
2. If external speakers are attached, it is recommended that VOLUME be set to 100, to ensure that the maximum possible sound is available to the speakers. The user can then decrease the volume if desired by using the control knobs on the speakers or the amplifier. If this argument value specifies anything less than 100, then the full sound volume will not be available to the speakers even if the control knobs are set to the maximum volume level.

BALANCE is an integer value ranging from 0 to 100 indicating the relative amount of sound to come from each speaker when an audio file is being played. Zero shuts off the right speaker entirely. A value from 1 to 49 causes more sound to come from the left speaker than the right one. A value of 50 causes sound to come equally from both speakers. A value from 51 to 99 favors the right speaker; 100 shuts off the left speaker entirely. If this argument is set to **AUD_DefaultBalance**, then a value of 50 is used.

Note: If the built-in workstation speaker is the only speaker, then for best results set BALANCE to 0 (left speaker only), and set VOLUME to 100.

BUSY_ACTION is an integer variable specifying what action to take if an audio file is already playing or being recorded when your program calls this sub-routine. Valid values are:

Constant	Meaning
AUD_Interrupt	Interrupt the current audio action and start playing immediately.
AUD_Return	Do not interrupt the current audio action. Ignore this request to play and return an errorlevel value of 4.
AUD_Queue	Queue this request to play until the ACPA becomes available. Note: If you select this option and the playback request is queued, any AUD_Play AUD_OpenError event notification that results is signalled to your program at the time the playback request is taken off the queue for processing, not at the time it is put on the queue.

18.2.7.2 Call

A call to **AUDPlay()** is made according to the following syntax.

Model

```
call AUDPlay( AUDIO_FILENAME,  
              PLAY_START_POSITION,  
              PLAY_STOP_POSITION,  
              VOLUME,  
              BALANCE,  
              BUSY_ACTION )
```

Returns:	errorlevel	Error Integer Constant/Explanation
	0	AUD_NORMAL_COMPLETION Normal completion.
	4	AUD_ERR_BUSY EASEL OS/2 EE audio support is busy and a value of AUD_Return was specified for the BUSY_ACTION argument. The request is not honored.
	16	AUD_ERR_QUEUE_FULL A value of AUD_Queue was specified for the BUSY_ACTION argument, but the queue of requests to play and record is full.
	20	AUD_ERR_INVALID_PARMS Invalid argument list.

Example:

```
string Music is "BEET5" # Filename of the music file  
integer Zero is 0      # Value for start/stop times  
integer Volume is AUD_DefaultVolume # Set to default  
integer Balance is AUD_DefaultBalance # Set to default  
integer BusyAction is AUD_Interrupt # Interrupt if busy  
integer ErrorLevel  
:
```

```
response to start
  call AUDPlay( Music,
               Zero,
               Zero,
               Volume,
               Balance,
               BusyAction )
  copy errorlevel to ErrorLevel
  send "Return Code for call to AUDPlay is:" ErrorLevel "
    to errorlog
  exit
```


18.2.8 AUDQueryBusy()

The **AUDQueryBusy()** subroutine is called to query EASEL OS/2 EE audio support to determine whether an audio file is currently being played or recorded. The subroutine returns the current status of EASEL OS/2 EE audio support.

AUDQueryBusy() is declared in the EASEL OS/2 EE audio support include file (**eslaudio.inc**) according to the following syntax:

```
subroutine AUDQueryBusy( integer:BUSY_STATUS )
    library "eslaudio"
```

18.2.8.1 Arguments

BUSY_STATUS is an integer variable that receives a number representing current EASEL OS/2 EE audio support status. The error number can be interrogated using the integer constants listed below.

Constant	Meaning
AUD_NotBusy	EASEL OS/2 EE audio support is not busy.
AUD_BusyPlaying	Playback is in progress.
AUD_BusyRecord	Recording is in progress.
AUD_BusyPause	Playback or recording is in progress but paused.

18.2.8.2 Call

A call to **AUDQueryBusy()** is made according to the following syntax:

Model

```
call AUDQueryBusy( BUSY_STATUS )
```

Returns:	errorlevel	Explanation
	0	AUD_NORMAL_COMPLETION Normal completion.

Example:

```
string Music is "BEET5" # Filename of the music file
integer Status           # Receives current status
:
call AUDQueryBusy( Status )
if (Status = AUD_NotBusy ) then
    call AUDPlay( Music ... )
:
end if
```


18.2.10 AUDQueryPlayingTime()

The **AUDQueryPlayingTime()** subroutine is called to determine the total time required to play an audio file. This subroutine can specify either an inactive audio file or one currently being played.

AUDQueryPlayingTime() is declared in the EASEL OS/2 EE audio support include file (**eslaudio.inc**) according to the following syntax:

```
subroutine AUDQueryPlayingTime( string:AUDIO_FILENAME,
                                integer:PLAYING_TIME )
    library "eslaudio"
```

18.2.10.1 Arguments

AUDIO_FILENAME is a string variable that must contain the name of the audio file whose playing time you want to query; do not specify the file extension. Setting the argument to a null string value (" ") returns the total playing time of the file currently being played.

PLAYING_TIME is an integer variable that will be modified to contain the total playing time of an audio file in milliseconds. If the file cannot be found, PLAYING_TIME will be set to zero (0).

18.2.10.2 Call

A call to **AUDQueryPlayingTime()** is made according to the following syntax:

Model

```
call AUDQueryPlayingTime( AUDIO_FILENAME,
                           PLAYING_TIME )
```

Returns:	errorlevel	Error Integer Constant/Explanation
	0	AUD_NORMAL_COMPLETION Normal completion.
	4	AUD_ERR_NOT_BUSY The AUDIO_FILENAME argument was set to null, but no audio file is currently being played, recorded, or paused.
	8	AUD_ERR_OPEN Attempt to open the specified audio file failed; file playing time is set to zero.

18.2.11 AUDRecord()

The **AUDRecord()** subroutine is called to record a segment of sound. If you specify a new audio filename, **AUDRecord()** will create a new audio file set. If you specify an existing audio file name, **AUDRecord()** will record over the existing file without notification. To protect an audio file from being re-recorded, use the **AUDQueryPlayingTime()** subroutine to determine whether the file already exists before calling **AUDRecord()**.

Note: If the audio file you ask to record cannot be opened, an **AUD_Error AUD_OpenError** event notification will be signalled to your program; if sufficient disk space is not available to contain the new recording, an **AUD_Error AUD_DiskFull** event will be signalled. Your program should contain **response to stimulus** statements monitoring for these events.

AUDRecord() is declared in the EASEL OS/2 EE audio support include file (**esludio.inc**) according to the following syntax:

```
subroutine AUDRecord( string:AUDIO_FILENAME
                    integer:RECORD_START_POSITION,
                    integer:RECORD_STOP_POSITION,
                    integer:SOUND_SOURCE,
                    integer:SOUND_QUALITY,
                    integer:BEEP,
                    integer:BUSY_ACTION )

    library "esludio"
```

18.2.11.1 Arguments

AUDIO_FILENAME is a string variable that must contain the name of the audio file set to be created or re-recorded; do not specify the file extension. This argument cannot be null. Maximum length is eight characters. If you specify a new audio filename, EASEL OS/2 EE audio support will create two files with same name (taken from **AUDIO_FILENAME**) but with different extensions: a control file with an extension of **lau**, and a file containing the digitized representation of the sound with an extension of **lad**.

RECORD_START_POSITION is an integer value (expressed in milliseconds) indicating the position in the audio file at which recording should start. A zero value means start at the beginning. You can re-record a partial segment of an existing audio file by specifying a value that corresponds to the start position of the segment you want to record over.

Notes:

1. If you are recording a new file, it is recommended that you specify a zero value for `RECORD_START_POSITION`. If you specify a non-zero value, then you will not be able to use the “play from the beginning” value (zero) for `PLAY_START_POSITION`; instead, you will always have to use the same value for `PLAY_START_POSITION` that you specify here for `RECORD_START_POSITION`.
2. Note also that while re-recording does have the desired effect of inserting sound segments in the specified places, it does NOT reuse audio file space. Instead it enlarges the audio file by appending the re-recorded segments to the end. Therefore, continued re-recording can increase the size of an audio file considerably. Disk space is used more efficiently if you erase and then re-record an entire audio file, rather than re-record only segments of it.

`RECORD_STOP_POSITION` is an integer value (expressed in milliseconds) indicating the position in the audio file at which recording should stop. A zero value means that there is no specified stop position and that the **AUDStop()** subroutine must be used to stop the recording process.

`SOUND_SOURCE` is an integer value specifying the source of the sound. Valid values are:

AUD_Mike	Specifies that sound input will come from a microphone.
AUD_MikeLow	Specifies that input will come from a microphone with a low sensitivity. AUD_MikeLow will cause the ACPA to amplify the input signal more than for the AUD_Mike value.
AUD_Line	Specifies that input will come from a prerecorded source (for example compact disc, audio cassette tape, etc.).
AUD_LineLeft	Specifies that input will come from a prerecorded source, but only left channel input will be recorded.
AUD_LineRight	Specifies that input will come from a prerecorded source, but only right channel input will be recorded.

`SOUND_QUALITY` is an integer value specifying the desired quality of the recording, or in other words, the level of sound detail to be captured in the recording. The higher the quality, the larger the audio files will be. Valid values are:

AUD_VoiceQuality	Record at 5.5K bytes/second, considered “voice quality.” This value records in monophonic mode.
-------------------------	---

- AUD_MusicQuality** Record at 11K bytes/second, considered “music quality.” If this argument is set to **AUD_DefaultQuality**, then this is the value used. This value records in monophonic mode.
- AUD_StereoQuality** Record at 22K bytes/second, considered “stereo quality.” This value records in stereophonic mode.

BEEP is an integer value specifying whether EASEL OS/2 EE audio support should signal that recording is about to begin by sounding a beep. Valid values are:

- AUD_Beep** Causes a beep tone to sound just prior to the start of recording.
- AUD_NoBeep** Suppresses the beep. No signal is given prior to the start of recording.

BUSY_ACTION is an integer value specifying what action to take if an audio file is already playing or being recorded when your program calls this subroutine. Valid values are:

- AUD_Interrupt** Interrupt the current audio action and start recording immediately.
- AUD_Return** Do not interrupt the current audio action. Ignore this request to record and return an **errorlevel** value of 4.
- AUD_Queue** Queue this request to record until the ACPA becomes available.

18.2.11.2 Call

A call to **AUDRecord()** is made according to the following syntax:

Model

```
call AUDRecord( AUDIO_FILENAME,
                RECORD_START_POSITION,
                RECORD_STOP_POSITION,
                SOUND_SOURCE,
                SOUND_QUALITY,
                BEEP,
                BUSY_ACTION )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	AUD_NORMAL_COMPLETION Normal completion.

- 4 AUD_ERR_BUSY**
EASEL OS/2 EE audio support is busy and a value of **AUD_Return** was specified for the **BUSY_ACTION** argument. The request is not honored.
- 16 AUD_ERR_QUEUE_FULL**
A value of **AUD_Queue** was specified for the **BUSY_ACTION** argument, but the queue of requests to play and record is full.
- .20 AUD_ERR_INVALID_PARMS**
Invalid argument list.

Example:

```
string NewRecording is "audition"
integer Zero is 0    # Start at beginning
integer StopTime is 30000  # 30 seconds
integer Mike is AUD_MikeLow
integer Quality is AUD_DefaultQuality  # Music
integer Beep is AUD_Beep
integer BusyAction is AUD_Interrupt
:
call AUDRecord( NewRecording,
                Zero,
                StopTime,
                Mike,
                Quality,
                Beep,
                BusyAction )
```


18.2.12 AUDResume()

The **AUDResume()** subroutine is called to resume playing or recording an audio file that was originally started by **AUDPlay()** or **AUDRecord()**, and then suspended by **AUDPause()**. This subroutine has no arguments.

AUDResume() is declared in the EASEL OS/2 EE audio support include file (**esludio.inc**) according to the following syntax:

```
subroutine AUDResume()  
    library "esludio"
```

18.2.12.1 Arguments

None

18.2.12.2 Call

A call to **AUDResume()** is made according to the following syntax:

Model

call AUDResume()

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	AUD_NORMAL_COMPLETION Normal completion.
	4	AUD_ERR_ALREADY_DONE There was no paused audio file to resume.

18.2.13 AUDSetBalance()

The **AUDSetBalance()** subroutine is called to adjust the speaker balance level while the audio file is being played.

AUDSetBalance() is declared in the EASEL OS/2 EE audio support include file (**eslaudio.inc**) according to the following syntax:

```
subroutine AUDSetBalance( integer:BALANCE )  
    library "eslaudio"
```

18.2.13.1 Arguments

BALANCE is an integer value ranging from 0 to 100 indicating the relative amount of sound to come from each speaker when an audio file is being played. Zero shuts off the right speaker entirely. A value from 1 to 49 causes more sound to come from the left speaker than the right one. A value of 50 causes sound to come equally from both speakers. A value from 51 to 99 favors the right speaker; 100 shuts off the left speaker entirely. If this argument is set to **AUD_DefaultBalance**, then a value of 50 is used.

Note: If the built-in workstation speaker is the only speaker, then for best results set BALANCE to 0 (left speaker only).

18.2.13.2 Call

A call to **AUDSetBalance()** is made according to the following syntax:

Model

```
call AUDSetBalance( BALANCE )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	AUD_NORMAL_COMPLETION Normal completion.
	4	AUD_ERR_NOT_BUSY No audio file is being played, recorded, or paused.
	20	AUD_ERR_INVALID_PARMS Invalid argument list.

Example:

```

string Music is "BEET5" # Filename of the music file
integer Zero is 0        # Value for start/stop times
integer Volume is AUD_DefaultVolume # Default is 100
integer Balance is AUD_DefaultBalance # Default is 50
integer BusyAction is AUD_Interrupt # Interrupt if busy
integer ErrorLevel
:
response to start
  call AUDPlay( Music,
               Zero,
               Zero,
               Volume,
               Balance,
               BusyAction )
  copy errorlevel to ErrorLevel
  send "Return Code for call to AUDPlay is:" ErrorLevel "\n"
    to errorlog
  exit
:
# Allow adjustments for changes in equipment
response to item PCSpeakerOnly from EquipmentMenu
  copy 0 to Balance
  call AUDSetBalance( Balance )
  :

```

18.2.14 AUDSetVolume()

The **AUDSetVolume()** subroutine is called to adjust the volume of the sound while it is being played back.

AUDSetVolume() is declared in the EASEL OS/2 EE audio support include file (**eslaudio.inc**) according to the following syntax:

```
subroutine AUDSetVolume( integer:VOLUME )
    library "eslaudio"
```

18.2.14.1 Arguments

VOLUME is an integer value ranging from 0 to 100 indicating the relative volume at which to play back the file. If this argument is set to **AUD_DefaultVolume**, then a value of 100 is used.

Notes:

- 1. If the built-in workstation speaker is the only speaker, then for best results set VOLUME to 100.
- 2. If external speakers are attached, it is recommended that VOLUME be set to 100, to ensure that the maximum possible sound is available to the speakers. The user can then decrease the volume if desired by using the control knobs on the speakers or the amplifier. If this argument value specifies anything less than 100, then the full sound volume will not be available to the speakers even if the control knobs are set to the maximum volume level.

18.2.14.2 Call

A call to **AUDSetVolume()** is made according to the following syntax:

Model

```
call AUDSetVolume( VOLUME )
```

Returns:	errorlevel	Error Integer Constant/Explanation
	0	AUD_NORMAL_COMPLETION Normal completion.
	4	AUD_ERR_NOT_BUSY No audio file is being played, recorded, or paused.
	20	AUD_ERR_INVALID_PARMS Invalid argument list.

18.2.15 AUDStop()

The **AUDStop()** subroutine is called to stop the audio file currently being played or recorded. It can also be used to purge the entire queue of requests to play and/or record.

Note: If zero is specified for the **RECORD_STOP_POSITION** argument on the **AUDRecord()** subroutine, then **AUDStop()** must be used to stop the recording process.

AUDStop() is declared in the EASEL OS/2 EE audio support include file (**eslaudio.inc**) according to the following syntax:

```
subroutine AUDStop( integer:PURGE_ACTION )  
    library "eslaudio"
```

18.2.15.1 Arguments

PURGE_ACTION is an integer variable specifying whether the queue of requests to play and/or record should be purged. Valid values are:

AUD_NoPurge	Stop only the audio file currently being played or recorded, and process the next request on the audio queue.
AUD_Purge	Stop the audio file currently being played or recorded, and also erase from the queue all pending requests to play or record.

18.2.15.2 Call

A call to **AUDStop()** is made according to the following syntax:

Model

```
call AUDStop( PURGE_ACTION )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	AUD_NORMAL_COMPLETION Normal completion.
	4	AUD_ERR_ALREADY_DONE EASEL OS/2 EE audio support has already stopped playing or recording the audio file.
	20	AUD_ERR_INVALID_PARMS Invalid argument list.

Example: integer PurgeAction is AUD_NoPurge # Don't purge queue
 :
 call AUDStop(PurgeAction)

18.3 Sample Program Using Audio Support

```
# This is a skeleton of an audio program that prompts a user for
# a voice recording. The user is first prompted with a voice
# recording that says "Enter your message at the beep". The record
# function then generates the beep and records. Audio errors are
# checked in the AUDIO_EVENT stimulus response block.

include "message.inc"           # ReplytoMessage include file
include "eslaudio.inc"         # Audio include file
stimulus AUDIO_EVENT "eslaudio.dll" # AUDIO_EVENT is a name for the
                                   # set of messages/stimuli
                                   # generated by the audio dll

string  AudioMessage
string  AudioPromptFileName
string  AudioRecordFileName
integer Balance is AUD_DefaultBalance
integer Beep is AUD_Beep
integer Interrupt is AUD_Interrupt
integer MikeInput is AUD_Mike
string  NullString is ""
integer Queue is AUD_Queue
integer TenSeconds is 10000
integer VoiceQuality is AUD_VoiceQuality
integer Volume is AUD_DefaultVolume
integer Zero is 0

# EASEL OS/2 EE program initialization
response to start

    call AUDOpen( NullString )    # Open the audio API;
                                   # audio files in DPATH

    # if open was unsuccessful then tell the user and exit
    if ( errorlevel != 0 ) then

        # display message box if not successful
        copy ReplytoMessage( "Audio Sample",
                              "Can't Open Audio!",
                              MessageOK,
                              1,
                              MessageIconHand ) to AudioMessage

        exit
    end if

# response to a Record pushbutton
response to Record in AudioDialogBox
```

```

if ( AudioRecordFileName != "" and AudioPromptFilename != "" ) then

    # erase the audio file if it exists
    call AudErase( AudioRecordFileName )

    # play the record prompt
    call AUDPlay( AudioPromptFileName,    # name of audio prompt file
                  Zero, Zero,              # play the complete prompt
                  Volume, Balance,         # at this volume/balance
                  Interrupt )              # interrupt any other audio

    # queue the audio record request after the record prompt
    call AUDRecord( AudioRecordFileName,   # name of file to record
                   Zero, TenSeconds,       # record for 10 sec max
                   MikeInput,              # take input from microphone
                   VoiceQuality,           # use voice quality
                   Beep,                   # beep before recording
                   Queue )                 # queue this record request
                                         # after initiating prompt play

end if

# response to a Cancel pushbutton
response to Cancel in AudioDialogBox

# stop any play/record in progress and purge the audio queue
call AUDStop( NullString )

# close the audio API
call AUDClose()
exit

# response to audio errors
response to stimulus AUDIO_EVENT AUD_Error

copy "Fatal Audio Error!" to AudioMessage

if ( eventparam = AUD_OpenError ) then
    copy "Can't Open Audio File!" to AudioMessage
end if

if ( eventparam = AUD_DiskFull ) then
    copy "Disk Write Error!" to AudioMessage
end if

copy ReplyToMessage( "Audio Sample",
                    AudioMessage,
                    MessageOK,
                    1,
                    MessageIconHand ) to AudioMessage

```

18.4 Audio Support Reserved Constants

The following reserved constants are used in EASEL OS/2 EE audio support. Be sure that your program does not contain any of these identifiers except when using them in the context of EASEL OS/2 EE audio support.

EASEL OS/2 EE audio support integer constants are used

- To set subroutine argument values
- As stimulus event filters
- As stimulus argument filters

AUD_Beep	AUD_Mike
AUD_BusyPausing	AUD_Mikelow
AUD_BusyPlaying	AUD_Music
AUD_BusyRecording	AUD_NoBeep
AUD_DefaultBalance	AUD_NoPurge
AUD_DefaultQuality	AUD_NotBusy
AUD_DefaultVolume	AUD_OpenError
AUD_DefaultQuality	AUD_Play
AUD_DiskFull	AUD_Purge
AUD_End	AUD_Queue
AUD_Error	AUD_Return
AUD_Interrupt	AUD_Record
AUD_Line	AUD_Start
AUD_LineLeft	AUD_StereoQuality
AUD_LineRight	AUD_VoiceQuality

AUDQueryBusy Return Values

AUD_BusyPausing
AUD_BusyPlaying
AUD_BusyRecording
AUD_Notbusy

EASEL OS/2 EE audio support Return Codes

AUD_ERR_ALREADY_DONE
AUD_ERR_BUSY
AUD_ERR_INVALID_PARMS
AUD_ERR_NOT_BUSY
AUD_ERR_OPEN
AUD_ERR_QUEUE_FULL
AUD_NORMAL_COMPLETION

Chapter 19. APPC Support

19.1 Overview

Advanced Program-to-Program Communication (APPC) is the OS/2 Extended Edition implementation of the Systems Network Architecture (SNA) LU 6.2 protocol that allows programs in different computers to communicate. EASEL OS/2 EE APPC support, which uses APPC support in the OS/2 Extended Edition Communications Manager, allows an EASEL OS/2 EE program to talk to a partner program (EASEL OS/2 EE or non-EASEL OS/2 EE) in another machine. The other machine can be a System/370™ or AS/400™ host, or another IBM OS/2 Extended Edition programmable workstation. EASEL OS/2 EE APPC offers a simplified interface to the OS/2 Extended Edition APPC application program interface (API), and provides support for half-duplex, mapped conversations.

Half-duplex conversations are the type supported by OS/2 Extended Edition APPC. In that type of conversation, at any given time only one partner has permission to send data; the other partner must be ready to receive. The sending and receiving partner programs can optionally switch roles, with mutual agreement and coordination.

If your EASEL OS/2 EE application needs to conduct a *full-duplex* conversation which allows the same program to send and receive simultaneously, you can simulate full-duplex capability by conducting two half-duplex conversations within one EASEL OS/2 EE APPC program.

Using EASEL OS/2 EE APPC support, an EASEL OS/2 EE program can either initiate a conversation with a partner program or accept an initiation request from the partner. The EASEL OS/2 EE APPC program can be written to always initiate the conversation, to always accept an initiation request, or to dynamically decide which role to play—initiator or acceptor—based on a parameter passed to the EASEL OS/2 EE program when it is started.

The EASEL OS/2 EE profiles contain configuration values set and used by EASEL OS/2 EE. Many of the values are identical to ones set during OS/2 Extended Edition APPC configuration. You must create profile information before attempting to use EASEL OS/2 EE APPC support the first time, and maintain it thereafter. Subroutines that you can use to set and query profile values are provided to help you do this. A sample EASEL OS/2 EE program, APPCPROF.EAL, to do profile maintenance is also provided.

System/370 and AS/400 are trademarks of International Business Machines Corporation.

EASEL OS/2 EE APPC support consists of two transaction programs (TPs), and subroutines resident in an EASEL OS/2 EE dynamic link library (DLL). The TPs are described in more detail in 19.1.8, “How EASEL OS/2 EE APPC Support Works” on page 19-6.

In your EASEL OS/2 EE program, calls are made to the subroutines to:

- Start a conversation
- Send data
- Receive data
- Get conversation information
- End a conversation

Responses from the EASEL OS/2 EE APPC DLL, in the form of messages, are returned (via the EASEL OS/2 EE Runtime Facility) to your EASEL OS/2 EE program where they trigger response definitions that pertain to APPC events. For example, the following response statement:

response to stimulus APPC_EVENT APPC_Data

and its action statements would execute when the EASEL OS/2 EE APPC DLL signalled your program that data from the APPC conversation had become available.

19.1.1 The Automatic Interface

There are two levels of EASEL OS/2 EE APPC interface: *automatic* and *controlled*. Your EASEL OS/2 EE APPC program indicates which interface it will be using when it starts a conversation.

Using the automatic interface cuts down on the number of EASEL OS/2 EE APPC subroutine calls you need to make in your EASEL OS/2 EE program to send, receive, or exchange data, and therefore reduces your program's complexity. With this level of support, EASEL OS/2 EE performs certain required APPC functions for your program automatically, such as switching the conversation state. The underlying OS/2 Extended Edition APPC architecture defines a set of *states* that a conversation can be in; for example, *Send* state or *Receive* state. (Conversation state is judged from your program's point of view.) The conversation must be in *Send* state in order for your program to send data, and in *Receive* state for it to receive.

EASEL OS/2 EE's automatic interface changes the conversation state automatically in the following situations, eliminating the need for you to change the state explicitly via an EASEL OS/2 EE APPC subroutine call:

- When your program indicates it has completed sending data, or that it is ready to receive data.
- When your program, after receiving data, issues an **APPCSendString()** subroutine call.

In this situation, because the partner currently has permission to send, a request to send will be issued and the data to be sent will be queued

until permission is granted. When authority to send is received, the queued data will be sent. Note that both partners must cooperate in relinquishing authority to send. A program may continue to send data as long as it wishes, and the receiving partner must be prepared to process this data for the duration.

- When your program remains inactive for a specified period of time after sending data and the partner program, after receiving data, indicates that it wants to send. The state will be switched automatically as long as all your program's requests to send data have been satisfied.

Automatic state switching can be disabled in either of two ways:

- At program startup (see the `STATE_SWITCH` argument in 19.5, "Detailed Subroutine Descriptions" on page 19-26 and 19.5.24, "`APPCStart( )`" on page 19-88)
- By an EASEL OS/2 EE APPC profile value (see the `INACTIVITY_TIMEOUT` argument set in 19.5.23, "`APPCSetInitProfile( )`" on page 19-84 and 19.5.22, "`APPCSetAcceptProfile( )`" on page 19-81).

19.1.2 The Controlled Interface

The controlled interface is available for you to use if you are an experienced APPC programmer and need full APPC flexibility. There are cases when an application needs more control over APPC state transitions; in those cases, automatic state switching would increase the complexity of the problem it is trying to solve. Using the controlled interface means you will have to make more subroutine calls than you would if you used the automatic interface, but you gain precise control over the APPC conversation.

19.1.3 Responding to APPC Events

Your EASEL OS/2 EE program is notified via the **response to stimulus** mechanism when APPC events occur. You can write a response definition for each event, or else write responses for some events and ignore others.

*There are three APPC events your program must not ignore—that is, for which you must write a **response to stimulus** statement and appropriate action statements. They are:*

1. Data has arrived from your partner program.
2. "End of conversation" has been signalled by your partner.
3. A system error has occurred.

Responding to the other APPC events is optional. Respond to the ones that apply to your application:

- Start of a partner application.
- Partner switched from sending to receiving.
- Partner wants to send data.
- Confirmation from partner received.

- Confirmation requested.
- Partner reports an error.

For more information on EASEL OS/2 EE APPC event processing and the complete list of event notifications, see 19.2, “EASEL OS/2 EE APPC Events” on page 19-8.

19.1.4 APPC Dynamic Link Library

The EASEL OS/2 EE APPC dynamic link library must be identified in your program as a stimulus DLL—one that will send messages to the EASEL OS/2 EE Runtime Facility.

Your EASEL OS/2 EE program must contain the following line:

```
stimulus APPC_EVENT "eslappc.dll"
```

where APPC_EVENT is any valid EASEL OS/2 EE identifier you have chosen to denote messages that represent APPC events.

19.1.5 Include File

The EASEL OS/2 EE APPC DLL has a corresponding include file that you must reference in your EASEL OS/2 EE program before you can invoke any of the APPC subroutines in the DLL.

Your EASEL OS/2 EE program must contain the following line:

```
include "eslappc.inc"
```

The include file declares the EASEL OS/2 EE APPC subroutines and argument types. It also contains definitions for EASEL OS/2 EE APPC integer constants, which are used both to set variables for some subroutine calls and as identifiers for EASEL OS/2 EE APPC event notifications.

19.1.6 Confirmation Processing

EASEL OS/2 EE APPC gives you the option of enabling *confirmation processing* for the APPC conversation, which makes it possible for your program and its partner to exchange notifications that data has been received and processed successfully. Your EASEL OS/2 EE program optionally enables confirmation processing when it starts the conversation (see the SYNC_LEVEL argument on the **APPCstart()** subroutine). In addition to enabling confirmation processing at program startup, your program must also request and send confirmation notifications by calling the appropriate subroutines. Any partner program that wants to communicate with an EASEL OS/2 EE APPC program that has requested confirmation processing must also be started with confirmation processing enabled.

19.1.7 Forcing a Send

Calling the **APPCSendString()** subroutine causes the data being sent to accumulate in a buffer. Normally, the data will not actually be transmitted until the buffer is full or state switching occurs causing the buffer to be flushed, but the **APPCSendString()** subroutine has an option that allows you to force the data to be transmitted immediately.

19.1.8 How EASEL OS/2 EE APPC Support Works

EASEL OS/2 EE APPC support consists of the following:

ESLAPPCI.EXE	An initiating transaction program (TP)
ESLAPPCA.EXE	A transaction program that accepts an initiation request
ESLAPPC.DLL	An OS/2 Extended Edition dynamic link library containing EASEL OS/2 EE APPC externalized subroutines and common routines

The following example describes how an APPC conversation between two hypothetical EASEL OS/2 EE APPC programs—PROGRAM1 and PROGRAM2—on different machines, would be established and would proceed. Both programs have been designed to be capable of both sending and receiving. The example illustrates use of the automatic interface.

The EASEL OS/2 EE Runtime Facility and PROGRAM1 are begun (the EASEL OS/2 EE APPC profile already exists, and PROGRAM1 uses it to determine what program to invoke at the remote machine). To initiate the APPC conversation, PROGRAM1 calls the **APPCStart()** subroutine.

APPCStart() obtains storage for a conversation control block, starts the ESLAPPCI.EXE transaction program as a separate process, and returns a conversation identifier—a handle—to PROGRAM1.

The ESLAPPCI.EXE TP operates asynchronously to the EASEL OS/2 EE Runtime Facility process. ESLAPPCI.EXE issues the OS/2 Extended Edition APPC verbs TP_STARTED and MC_ALLOCATE to start the conversation. If a startup error occurs at this point, ESLAPPCI.EXE indicates it by POSTing a message to the EASEL OS/2 EE.

The MC_ALLOCATE verb triggers OS/2 Extended Edition Communications Manager in the remote machine to activate the ESLAPPCA.EXE TP (note that ESLAPPCA.EXE may also be started by the user at the remote machine, and would simply wait for the initiation request). When ESLAPPCA.EXE is started, the EASEL OS/2 EE Runtime Facility and PROGRAM2 are presumed not to be running. One or more EASEL OS/2 EE programs unrelated to APPC communications (along with one or more copies of the EASEL OS/2 EE) may in fact already be running at the remote machine. However, it is only the specific EASEL OS/2 EE program(s) configured to participate in APPC conversations—PROGRAM2 in this case—that the ESLAPPCA.EXE will attempt to start.

The ESLAPPCA.EXE TP issues a RECEIVE_ALLOCATE APPC verb. If that is successful, ESLAPPCA.EXE gets a conversation block and then looks at the EASEL OS/2 EE APPC profile to get the invocation command needed to start the EASEL OS/2 EE Runtime Facility and PROGRAM2. DosExecPgm is used to execute the invocation command. When PROGRAM2 gets control it must call the **APPCAcceptStart()** subroutine to get the conversation ID. If the EASEL OS/2 EE Runtime Facility and the EASEL OS/2 EE partner program are not successfully started, ESLAPPCA.EXE issues an APPC MC_DEALLOCATE verb with the ABEND indication.

Unless an error notification has been received, the conversation between the two EASEL OS/2 EE programs can now proceed. Keep in mind that the APPC protocol allows only one program at a time to be in *Send* state, and while it is in *Send* state, its partner must be in *Receive* state.

PROGRAM1 calls the **APPCSendString()** subroutine, and a “request to send” is queued to the ESLAPPCI.EXE TP running locally on its machine. The TP then checks whether the conversation is in *Send* state. If so, the TP issues the MC_SEND_DATA APPC verb, and the data is transmitted (immediately or eventually, depending on whether the data was forced).

If the conversation is in *Receive* state, the ESLAPPCI.EXE TP issues an MC_REQUEST_TO_SEND APPC verb followed by an MC_RECEIVE_AND_POST. Its partner TP, ESLAPPCA.EXE, which by implication currently has *Send* state, then receives the “request to send” notification as a response to its next MC_SEND_DATA request.

When the partner TP, ESLAPPCA.EXE, has been inactive for a certain amount of time after data was last sent, an internal timer goes off causing ESLAPPCA.EXE to issue an MC_TEST_RTS verb to check whether a “request to send” is pending. When it receives the “request to send” from the ESLAPPCI.EXE TP, ESLAPPCA.EXE completes sending any queued data and then issues an MC_RECEIVE_AND_POST verb. When data from ESLAPPCI.EXE arrives, the ESLAPPCA.EXE TP places it in a local queue and POSTs a message to the EASEL OS/2 EE runtime, which in turn triggers the **response to stimulus ... APPC_Data** definition in PROGRAM2. When PROGRAM2 calls the **APPCGetString()** subroutine, the data is placed in an EASEL OS/2 EE string variable. An additional message is POSTed for each complete data buffer received.

Note: The length-of-inactivity timer value is specified in the EASEL OS/2 EE APPC profile. The timer is employed only in the automatic interface, and even then its use is optional. The timer value can be set to 0 to prevent EASEL OS/2 EE APPC from switching the conversation state automatically. See the INACTIVITY_TIMEOUT argument in 19.5.23, “APPCSetInitProfile()” on page 19-84 and 19.5.22, “APPCSetAcceptProfile()” on page 19-81 for more information.

19.2 EASEL OS/2 EE APPC Events

There are twelve different event notifications or stimuli that the EASEL OS/2 EE APPC DLL can generate. You respond to any of these events using the EASEL OS/2 EE **response to stimulus** statement, in the following model.

Model

```
response to stimulus APPC_EVENT [ APPC_Started |
                                APPC_Ended |
                                APPC_Data |
                                APPC_ConfirmRequest |
                                APPC_Confirmed |
                                APPC_RequestToSend |
                                APPC_SendState |
                                APPC_ReceiveError |
                                APPC_SendError |
                                APPC_SystemError |
                                APPC_InputReceived |
                                APPC_EndReceive ]
                                [CONVERSATION_ID]
```

In the above model, APPC_EVENT can be any valid EASEL OS/2 EE identifier that you have assigned in the stimulus declaration to denote an incoming message from the EASEL OS/2 EE APPC DLL (see 19.1.5, "Include File" on page 19-5). CONVERSATION_ID is an integer variable of your choice that contains the conversation identifier, a handle, returned from **APPCStart()** or **APPCAcceptStart()**.

The stimulus events listed in the above model are EASEL OS/2 EE integer constants, predefined for use in EASEL OS/2 EE APPC support. These constants are optionally used as "filtering" values on the **response to stimulus** statement to limit a given response definition to a specific APPC event. If you code an APPC **response to stimulus** statement without specifying a stimulus event, then all APPC events associated with the DLL specified in the stimulus declaration will trigger that response statement.

The meanings of the EASEL OS/2 EE APPC integer constants, when used as stimulus events, are listed below.

APPC_Started	The partner program has started. This event notification is useful if your program is a "service" application waiting for an initiator.
---------------------	---

APPC_Ended	The partner program has ended. Call the APPCEnd() subroutine to free the conversation resources.
APPC_Data	Data has arrived for this conversation from its partner. Call the APPCGetString() subroutine to get the information.
APPC_ConfirmRequest	The partner program requests confirmation. Your program should process the data, then indicate successful processing by sending a confirmation or unsuccessful processing by sending an error response.
APPC_Confirmed	The partner program confirms data has been received and processed successfully. This notification occurs when your program requested a confirmation on the last send.
APPC_RequestToSend	The partner program wants to send data.
APPC_SendState	The partner program has completed sending data and is ready to receive a reply. Your program has been granted authority to send. This notification can be taken to mean that all the data requested from the partner has been sent.
APPC_ReceiveError	An error was detected by the partner program in the data that it received.
APPC_SendError	An error has been detected by the partner program. The partner has a problem processing; it could not produce the data it wanted to send.
APPC_SystemError	A serious error has been detected. This error may be signalled by either EASEL OS/2 EE APPC support or an OS/2 Extended Edition APPC service. Call the APPCGetSystemError() subroutine to get detailed information about the error. The information should then be displayed or logged; it will help a person knowledgeable in communications diagnose the problem. Some errors may be able to be handled by the EASEL OS/2 EE APPC program using the controlled interface.
APPC_InputReceived	This is used in the controlled interface to indicate the start of a group of notifications connected to one call to APPCReceiveNotify() .
APPC_EndReceive	This is used in the controlled interface to indicate the end of a group of notifications connected to one call to APPCReceiveNotify() .

Some EASEL OS/2 EE APPC integer constants can also be used to set variable values for subroutine calls.

19.3 EASEL OS/2 EE APPC Subroutines

EASEL OS/2 EE APPC support consists of 25 subroutines callable by the EASEL OS/2 EE programmer.

Subroutines for Profile Access

APPCSetInitProfile()	APPCCSetAcceptProfile()
APPCCGetInitProfile()	APPCCGetAcceptProfile()
APPCCGetInitProfileNames()	APPCCGetAcceptProfileNames()

Subroutines to Send/Receive String Data

APPCCStart()	APPCCGetString()
APPCCAcceptStart()	APPCCGetSystemError()
APPCCSendString()	APPCCFlush()
APPCCReadyToReceive()	APPCCEnd()

Subroutines to Send/Receive Non-string Data

APPCCGetDLLData()
APPCCSendDLLData()

Subroutines for Confirmation Processing

APPCCRequestConfirm()
APPCCSendConfirmed()
APPCCSendError()

Subroutines to Switch Conversation State

APPCCReceiveNotify()
APPCCRequestToSend()
APPCCTestReqToSend()

Utility Subroutines

APPCCConvertA2E()
APPCCConvertE2A()
APPCCGetInfo()

19.3.1 Setting/Querying Profile Information

EASEL OS/2 EE APPC subroutines are provided for you to both query and update EASEL OS/2 EE APPC profile information. Most of the EASEL OS/2 EE APPC profile information required is the same as that required to configure APPC in the OS/2 Extended Edition Communications Manager profiles, and can be obtained from the same sources. EASEL OS/2 EE APPC profile information is used by the two EASEL OS/2 EE APPC TPs (ESLAPPCI.EXE and ESLAPPCA.EXE) to establish the EASEL OS/2 EE APPC session. Profile information used by the initiator TP (ESLAPPCI.EXE) is stored in the file ESLAPPCI.PRO; acceptor TP profile information is stored in ESLAPPCA.PRO. Both these files must reside in a directory defined in the OS/2 DPATH or PATH environment variable. Multiple profiles, for example representing different APPC configurations, can be put into each profile file.

Typically EASEL OS/2 EE APPC profile information is created once, then used repeatedly. If the EASEL OS/2 EE program you write to do APPC communications is an initiator, then at least one initiator profile must exist in the ESLAPPCI.PRO file before your program attempts to initiate its first APPC session; similarly, if your EASEL OS/2 EE program is an acceptor, then at least one acceptor profile must exist in the ESLAPPCA.PRO file. If your program both initiates and accepts, then both types of profile information must be present. The EASEL OS/2 EE program you write to do APPC communications can also create or update the profile information, or you can write a separate EASEL OS/2 EE program that does only profile maintenance. A sample program, APPCPROF.EAL, that does only profile maintenance, is provided.

There are two sets of profile subroutines. **APPCSetInitProfile()**, **APPCGetInitProfile()**, and **APPCGetInitProfileNames()** query and maintain profiles used by EASEL OS/2 EE APPC programs that initiate APPC conversations. **APPCSetAcceptProfile()**, **APPCGetAcceptProfile()**, and **APPCGetAcceptProfileNames()** query and maintain profiles used by acceptor EASEL OS/2 EE APPC programs (programs that accept initiation requests).

19.3.1.1 EASEL OS/2 EE APPC Profile Subroutines

```
call APPCSetInitProfile      ( PROFILE_NAME,  
                              ACTION,  
                              LOCAL_LU_ALIAS_NAME,  
                              PARTNER_LU_ALIAS_NAME,  
                              PARTNER_TP_NAME,  
                              MODE_NAME,  
                              SECURITY_PARAMETERS,  
                              INACTIVITY_TIMEOUT,  
                              BUFFER_SIZE,  
                              ASCII_TABLE,  
                              EBCDIC_TABLE )  
  
call APPCGetInitProfile     ( PROFILE_NAME,  
                              LOCAL_LU_ALIAS_NAME,  
                              PARTNER_LU_ALIAS_NAME,  
                              PARTNER_TP_NAME,  
                              MODE_NAME,  
                              SECURITY_PARAMETERS,  
                              INACTIVITY_TIMEOUT,  
                              BUFFER_SIZE,  
                              ASCII_TABLE,  
                              EBCDIC_TABLE )  
  
call APPCGetInitProfileNames ( PROFILE_NAMES )  
  
call APPCSetAcceptProfile   ( PROFILE_NAME,  
                              ACTION,  
                              INVOCATION_CMD,  
                              INACTIVITY_TIMEOUT,  
                              BUFFER_SIZE,  
                              ASCII_TABLE,  
                              EBCDIC_TABLE )  
  
call APPCGetAcceptProfile   ( PROFILE_NAME,  
                              INVOCATION_CMD,  
                              INACTIVITY_TIMEOUT,  
                              BUFFER_SIZE,  
                              ASCII_TABLE,  
                              EBCDIC_TABLE )  
  
call APPCGetAcceptProfileNames ( PROFILE_NAMES )
```

19.3.2 The Automatic Interface

The automatic interface subroutines are used to conduct an APPC conversation in which EASEL OS/2 EE APPC will automatically switch the conversation state when appropriate to do so.

All EASEL OS/2 EE APPC subroutines can be used in both the automatic interface and the controlled interface with the exception of **APPCReceiveNotify()**. This subroutine is used only in the controlled interface. For information about the controlled interface and the alternative level of support that it offers, see 19.3.5, “The Controlled Interface” on page 19-17.

Your EASEL OS/2 EE APPC program indicates when it starts a conversation which style of interface it will be using. Automatic state switching is enabled by calling the **APPCStart()** and **APPCAcceptStart()** subroutines with the STATE_SWITCH argument set to **APPC_Automatic**. Automatic state switching can be disabled in either of two ways:

1. By calling the **APPCStart()** and **APPCAcceptStart()** subroutines with the STATE_SWITCH argument set to **APPC_Controlled**, or
2. By setting the INACTIVITY_TIMEOUT argument in the EASEL OS/2 EE APPC profile to zero (regardless of the setting of STATE_SWITCH). See 19.5.23, “APPCSetInitProfile()” on page 19-84 and 19.5.22, “APPCSetAcceptProfile()” on page 19-81.

Your program can voluntarily switch from *Send* state to *Receive* state either by:

- Using **APPC_ReadyToReceive** as the value for the MORE_TO_SEND argument on the **APPCSendString()** or **APPCSendDLLData()** subroutine calls, or
- Calling the **APPCReadyToReceive()** subroutine.

If you are using confirmation processing in the automatic interface, you should issue the request for confirmation from the partner as a parameter on the **APPCSend...** subroutines, or else issue a separate call to **APPCRequestConfirm()** IMMEDIATELY after doing the send. Otherwise the conversation state may switch from *Send* to *Receive*, and once in *Receive* state your attempt to send a request for confirmation would be rejected as an error. You should also monitor for **APPC_RequestToSend** event notifications to know that the conversation state has switched.

Note: If the partner issues a “request to send,” the conversation state for your program will change from *Send* to *Receive* if there is nothing pending to send and the inactivity timer has timed out. Automatic state switching will not occur if the timer value is set to zero. See 19.5.23, “APPCSetInitProfile()” on page 19-84 and 19.5.22, “APPCSetAcceptProfile()” on page 19-81.

19.3.2.1 EASEL OS/2 EE APPC Subroutines to Send/Receive String Data

```
call APPCStart          ( CONVERSATION_ID,
                        PROFILE_NAME,
                        SYNC_LEVEL,
                        STATE_SWITCH )

call APPCAcceptStart    ( CONVERSATION_ID,
                        PROFILE_NAME,
                        STATE_SWITCH )

call APPCSendString     ( CONVERSATION_ID,
                        STRING_VAR,
                        AUTO_CONVERT,
                        SEND_IND,
                        MORE_TO_SEND )

call APPCReadyToReceive ( CONVERSATION_ID )

call APPCGetString     ( CONVERSATION_ID,
                        STRING_VAR,
                        AUTO_CONVERT )

call APPCGetSystemError ( CONVERSATION_ID,
                        WORK_REQUEST_ERROR,
                        WORK_REQUEST_IDENTIFIER,
                        APPC_VERB,
                        PRIMARY_ERROR_CODE,
                        SECONDARY_ERROR_CODE )

call APPCFlush         ( CONVERSATION_ID )

call APPCEnd           ( CONVERSATION_ID,
                        SYNC_LEVEL )
```

19.3.3 Sending/Receiving Non-string Data

An EASEL OS/2 EE program can send and receive only string data. To enable other types of data to be sent and received, two EASEL OS/2 EE APPC subroutines are provided that can be called from user-written subroutines embedded in a user-written dynamic link library (DLL). The EASEL OS/2 EE APPC DLL routines can be used with either the automatic interface or the controlled interface. The user-written subroutines must understand the format of the data and must reformat the data into strings in order to send the data to or receive it from an EASEL OS/2 EE program.

To send non-string data in an APPC conversation, your EASEL OS/2 EE program would first call a user-written send routine (in its own DLL) rather than **APPCSendString()** to build the data buffer, doing the necessary string-

to-non-string conversion. The user-written send routine would then call **APPCSendDLLData()** to issue the APPC commands to do the send.

Similarly, to receive non-string data, your EASEL OS/2 EE program would call a user-written get routine instead of **APPCGetString()**. The user-written routine would call **APPCGetDLLData()** to get a pointer to the received data, convert it to string data, and then return it to your EASEL OS/2 EE program.

19.3.3.1 EASEL OS/2 EE APPC Subroutines to Send/Receive Non-string Data

```
APPCGetDLLData ( CONVERSATION_ID,  
                  BUFFER_POINTER,  
                  BUFFER_LENGTH );
```

```
APPCSendDLLData ( CONVERSATION_ID,  
                  BUFFER_POINTER,  
                  BUFFER_LENGTH,  
                  SENDIND,  
                  MORETOSEND );
```

19.3.4 Confirmation Processing

19.3.4.1 Synchronizing Processing

You may want your program to ensure that its partner has successfully processed all of the records sent to it before more are sent, or before continuing processing. This is known as synchronizing the conversation. Data transmission establishes a useful synchronization point for the conversation; after data is sent, your program should do nothing more until the partner either confirms that it has completed the work without error or else sends an error indication.

An APPC program (EASEL OS/2 EE or non-EASEL OS/2 EE) declares when it starts a conversation whether it supports and/or requests confirmation processing. Both partners must agree that this is an expected part of their protocol.

Your EASEL OS/2 EE APPC program can request confirmation of a send either when sending the last data item or else on a separate call that explicitly requests confirmation. If the partner replies positively, an **APPC_Confirmed** event notification is signalled in your program. If the partner replies with an error, an **APPC_ReceiveError** or an **APPC_SendError** event notification is signalled. Your program must contain response definitions for all these events.

The partner program can also request confirmation. Its request generates an **APPC_ConfirmRequest** event notification in your program. Your program

must include a response definition for this event, and should respond to the event by either sending a confirmation back to the partner or else by signalling an error. If your program ignores a confirmation request from its partner, the conversation will in effect be suspended, because the partner program will continue to wait for one of these responses.

Note: When data and a request for confirmation arrive simultaneously, the **APPC_Data** event will be signalled in your program before the **APPC_ConfirmRequest** event.

In detail, here is how to use EASEL OS/2 EE APPC confirmation processing:

1. Specify **APPC_Confirm** as the value for the **SYNC_LEVEL** argument on the **APPCStart()** subroutine call.
2. Request a send confirmation from your partner by either:
 - Specifying **APPC_Confirm** or **APPC_SyncLevel** as the value for the **SEND_IND** argument on the **APPCSendString()** or **APPCSendDLLData()** subroutine call
 - Calling the **APPCRequestConfirm()** subroutine. In the automatic interface, this call should immediately follow the last request that is to be confirmed.
3. Provide three response definitions in your program to trap any of three possible responses to the request for confirmation:
 - **response to stimulus ... APPC_Confirmed**
 - **response to stimulus ... APPC_ReceiveError**
 - **response to stimulus ... APPC_SendError**
4. To respond to a confirmation request from your partner, provide a **response to stimulus ... APPC_ConfirmRequest** response definition that calls either the **APPCSendConfirmed()** subroutine or the **APPCSendError()** subroutine.

19.3.4.2 Subroutines for Confirmation Processing

```
call APPCRequestConfirm ( CONVERSATION_ID )
```

```
call APPCSendConfirmed ( CONVERSATION_ID )
```

```
call APPCSendError      ( CONVERSATION_ID,  
                        ERROR_ORIGIN )
```

19.3.5 The Controlled Interface

19.3.5.1 Advanced Control Capabilities

The controlled interface is used to conduct an APPC conversation without automatic state switching—that is, one in which your EASEL OS/2 EE APPC program must explicitly request to receive and to send data. Using the controlled interface your program must also handle all error recovery for APPC system errors. All EASEL OS/2 EE APPC subroutines can be used in both the automatic interface and the controlled interface with the exception of **APPCReceiveNotify()**. This subroutine is used only in the controlled interface.

There are three basic conversation states that you need to understand in order to use the controlled interface: *Receive*, *Send*, and *Confirm*. For an introduction to the automatic interface, see 19.3.2, “The Automatic Interface” on page 19-14. The three conversation states are also briefly described below.

Knowledge of EASEL OS/2 EE APPC event notifications is assumed for the following discussion. See 19.2, “EASEL OS/2 EE APPC Events” on page 19-8.

Your EASEL OS/2 EE APPC program indicates that it will be using the controlled interface at conversation startup, by calling either **APPCStart()** or **APPCAcceptStart()** with the value for the `STATE_SWITCH` parameter set to **APPC_Controlled**. Once the conversation has begun, your program must keep track of the conversation state it is in, and explicitly control all switches in state. APPC errors will result if a subroutine is called in the wrong state. No attempt is made by EASEL OS/2 EE APPC to recover from APPC system errors; errors are simply signalled to your program as **APPC_SystemError** events. For simplicity, the rest of this discussion assumes that all subroutine calls complete without error.

19.3.5.2 Receive State

In *Receive* state your program is allowed to use the conversation to retrieve data and status information. To get the conversation into *Receive* state your program must issue one of the following calls:

- **APPCAcceptStart()** (which initializes the conversation)
- **APPCReceiveNotify()** (which requests data and/or status)
- **APPCReadyToReceive()**
- **APPCSendString()** with the value for the `MORE_TO_SEND` argument set to **APPC_ReadyToReceive**.

Once in *Receive* state your program calls **APPCReceiveNotify()** to request additional data or status notifications. One call to **APPCReceiveNotify()** can generate one or more notifications; **APPC_Data**, **APPC_ConfirmRequest**,

and **APPC_SendState** event notifications may all be sent to your program as a result of one call to **APPCReceiveNotify()**.

Event notifications are grouped, so you can determine when to issue the next call to **APPCReceiveNotify()**. The **APPC_InputReceived** notification arrives at your program first, followed by data or status notifications, and trailed by an **APPC_EndReceive** notification. An EASEL OS/2 EE response block designed to take advantage of this sequence is shown below.

The event notifications **APPC_InputReceived** and **APPC_EndReceive** bracket event notifications associated with one APPC receive verb or receive notify request, namely:

APPC_Data
APPC_ConfirmRequest
APPC_Ended
APPC_ReceiveError
APPC_SendError
APPC_SystemError
APPC_RequestToSend

Event notifications that can occur independent of the **APPC_InputReceived** and **APPC_EndReceive** signals are:

APPC_Started
APPC_Ended
APPC_Confirmed
APPC_ReceiveError
APPC_SendError
APPC_SystemError
APPC_RequestToSend

In *Receive* state your program can call:

- **APPCSendConfirmed()**
- **APPCReceiveNotify()** (which should be called every time an **APPC_EndReceive** event is signalled as long as more data or status information is still wanted)
- **APPCSendError()** (this call changes the conversation state to *Send*)
- **APPCRequestToSend()**,
- **APPCEnd()** (when **APPC_Abend** is specified).

19.3.5.3 Typical Event Notification Sequence

```
# Start of input
response to stimulus APPC_EVENT APPC_InputReceived
begin ProcessInput
    response to stimulus APPC_EVENT APPC_Data
        call APPCGetString(...)      # Get data
                                    # Process it
:

response to stimulus APPC_EVENT APPC_ConfirmRequest
    if (Process = "OK") then
        call APPCSendConfirmed(...)
    else
        call APPCSendError(...)      # Send error indication
        call APPCSendString(...)     # With APPC_ReadyToReceive
    end if                          # Give reason for error

response to stimulus APPC_EVENT APPC_SendState
    copy "Send" to State
    call APPCSendString(...)
    call APPCSendString(...)        # With APPC_Confirm

response to stimulus APPC_EVENT APPC_RequestToSend
    copy "Receive" to State

# Partner sent error
response to stimulus APPC_EVENT APPC_ReceiveError
    copy "Receive" to State
    copy "ReDo" to Process

# Partner sent error
response to stimulus APPC_EVENT APPC_SendError
    copy "Receive" to State
    copy "Error" to Process
response to stimulus APPC_EVENT APPC_SystemError
:
response to stimulus APPC_EVENT APPC_Ended
    if (Process = "OK") then
        call APPCEnd(...)
    end if

# End of input
response to stimulus APPC_EVENT APPC_EndReceive
    if (State = "Receive") then
        call APPCReceiveNotify(...) # Request more data
    end if
    leave block
end ProcessInput
```

19.3.5.4 Send State

In *Send* state your program has *send authority*; it is allowed to use the conversation to send data. To get the conversation into *Send* state your program must issue one of the following calls:

- **APPStart()** (which initializes the conversation)
- **APPRequestToSend()** (which requests *Send* state from the partner—this call must be followed by **APPReceiveNotify()** calls until the **APP_SendState** notification is received)
- **APPSendError** (to indicate a problem with the data that was received).

Once the conversation is in *Send* state your program may issue the following calls:

- **APPSendString()**
- **APPFlush()**
- **APPRequestConfirm()**
- **APPReceiveNotify()**
- **APPTestReqToSend()**
- **APPEnd()**

19.3.5.5 Confirm State

A request for confirmation from your partner program causes the conversation to go into *Confirm* state. Your program can recognize that the conversation has gone into *Confirm* state when it gets an **APP_ConfirmRequest** event notification. The partner will wait to receive a confirmation or an error response from your program. The only subroutines valid for your program to call when the conversation is in *Confirm* state are **APPSendConfirmed()** or **APPSendError()**.

19.3.5.6 The Controlled Interface Subroutines

Three subroutines are used primarily with the controlled interface:

APPReceiveNotify(), **APPRequestToSend()**, and **APPTestReqToSend()**. These three plus all the other EASEL OS/2 EE APPC subroutines together comprise the EASEL OS/2 EE APPC controlled interface.

19.3.5.7 Subroutines to Switch Conversation State

call APPReceiveNotify (CONVERSATION_ID)

call APPRequestToSend (CONVERSATION_ID)

call APPTestReqToSend (CONVERSATION_ID)

19.3.6 Utility Subroutines

Utility subroutines are provided to assist you in converting ASCII and EBCDIC strings, and also to retrieve information about an APPC conversation in progress.

Note that the **APPCConvertE2A()** subroutine would not need to be used if a value of **APPC_Convert** is used for the **AUTO_CONVERT** argument on the **APPCSendString()** call. **APPCConvertE2A()** typically is used to convert the EBCDIC information returned to the **APPCGetInfo()** subroutine into ASCII.

19.3.6.1 EASEL OS/2 EE APPC Utility Subroutines

```
call APPCConvertA2E ( CONVERSATION_ID,
                      STRING,
                      INDEX_START,
                      INDEX_END,
                      ASCII_TABLE,
                      EBCDIC_TABLE )

call APPCConvertE2A ( STRING,
                      INDEX_START,
                      INDEX_END,
                      ASCII_TABLE,
                      EBCDIC_TABLE )

call APPCGetInfo      ( CONVERSATION_ID,
                      APPC_TP_ID,
                      APPC_CONVERSATION_ID,
                      NET_NAME,
                      LOCAL_LU_NAME,
                      LOCAL_LU_ALIAS_NAME,
                      PARTNER_LU_ALIAS_NAME,
                      PARTNER_LU_UNINTERPRETED_NAME,
                      FULLYQUAL_PARTNER_LU_NAME,
                      PARTNER_TP_NAME,
                      MODE_NAME,
                      SYNC_LEVEL,
                      USER_ID )
```

19.4 Using APPC Subroutines

EASEL OS/2 EE APPC subroutines operate the same way as other EASEL OS/2 EE subroutines. To invoke an EASEL OS/2 EE APPC subroutine from your EASEL OS/2 EE program, use the **call** statement. The **call** statement syntax is the same for internal EASEL OS/2 EE subroutines and EASEL OS/2 EE APPC subroutines:

```
call SUB_NAME( ARG1[, ARG2]... )
```

where SUB_NAME is the identifier for a previously defined subroutine, and ARG1 [, ARG2]... are variables you have defined to contain the subroutine parameters. You must use variables as argument list parameters in an EASEL OS/2 EE APPC subroutine call; literals or constants cannot be used. Also, the type and number of arguments after the subroutine name must match the type and number of arguments in the declaration.

19.4.1 Setting Argument Values

The integer constants described in 19.2, "EASEL OS/2 EE APPC Events" on page 19-8 are pre-defined for use in EASEL OS/2 EE APPC support. They can be used to set values for subroutine calls by copying them into integer variables you have defined to be used as subroutine arguments. For those arguments that are integer variables and that have a default value—marked in this chapter with "(Default)"—you can indicate that the default value should be used by copying the integer constant **APPC_Default** into the argument. Of course, any other variables, constants, or literals can also be used to set subroutine argument values as long as the resulting value is valid for the argument. Note that subroutine arguments that are string variables cannot be defaulted; you must always supply a value.

The set of predefined integer constants you can use to set subroutine argument values is listed below:

Constant	Value	Where Used
APPC_Abend	11	APPCEnd()
APPC_Automatic	3	APPCStart() APPCAcceptStart()
APPC_BadGet	12	APPCSendError()
APPC_BadSend	13	APPCSendError()
APPC_Confirm	2	APPCStart() APPCSendString() APPCSendDLLData() APPCEnd()

Constant	Value	Where Used
APPC_Controlled	4	APPCStart() APPCAcceptStart()
APPC_Convert	6	APPCGetString() APPCSendString()
APPC_CreateProfile	14	APPCSetInitProfile() APPCSetAcceptProfile()
APPC_Default	0	All subroutines
APPC_DeleteProfile	16	APPCSetInitProfile() APPCSetAcceptProfile()
APPC_Flush	7	APPCSendString() APPCSendDLLData() APPCEnd()
APPC_MoreData	9	APPCSendString() APPCSendDLLData()
APPC_NoConvert	5	APPCGetString() APPCSendString()
APPC_None	1	APPCStart() APPCSendString() APPCSendDLLData()
APPC_ReadyToReceive	10	APPCSendString() APPCSendDLLData()
APPC_ReplaceProfile	15	APPCSetInitProfile() APPCSetAcceptProfile()
APPC_SyncLevel	8	APPCSendString() APPCSendDLLData() APPCEnd()

For the complete list of EASEL OS/2 EE APPC predefined integer constants, see 19.8.

19.4.2 Return Codes

If an EASEL OS/2 EE APPC subroutine encounters an error condition, the EASEL OS/2 EE built-in function **errorlevel** is set to the number corresponding to the error found. **errorlevel** will contain a value of zero (0) when the subroutine executes without error.

In your EASEL OS/2 EE program, you should refer to an error by its corresponding integer constant rather than its number; integer constants representing errors are listed under the *Returns* section of each subroutine description.

The EASEL OS/2 EE APPC return code values and integer constants are defined in the EASEL OS/2 EE APPC include file (**eslappc.inc**).

2

19.5 Detailed Subroutine Descriptions

19.5.1 APPCAcceptStart()

The **APPCAcceptStart()** subroutine is called to accept a conversation initiated by a program in another machine. An identifier (OS/2 handle) representing the conversation is returned. This identifier must be passed in the other APPC subroutine calls.

APPCAcceptStart() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCAcceptStart( integer:CONVERSATION_ID,  
                           string:PROFILE_NAME,  
                           integer:STATE_SWITCH )  
  
    library "eslappc"
```

19.5.1.1 Arguments

CONVERSATION_ID is an integer variable that is set to an identifier for the conversation by this subroutine.

PROFILE_NAME is a string variable that specifies the name of the profile created for the acceptor program. This name **MUST** be the same name specified as the remotely attachable TP name in the OS/2 Extended Edition Communications Manager profiles. This argument is case sensitive.

STATE_SWITCH is an integer variable used to indicate whether the automatic interface or the controlled interface is to be used. The automatic interface reduces the number of calls needed to send and receive data; state switching is automatic. The controlled interface provides full APPC flexibility to sophisticated applications. Valid constants are:

APPC_Automatic (Default) The conversation will use the automatic interface.

APPC_Controlled The conversation will use the controlled interface.

19.5.1.2 Call

A call to **APPCAcceptStart()** is made according to the following syntax:

Model

```
call APPCAcceptStart( CONVERSATION_ID,  
                      PROFILE_NAME,  
                      STATE_SWITCH )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	23	APPC_ERR_INV_STATE_SWITCH Invalid argument: STATE_SWITCH
	32	APPC_ERR_INV_PROFILE_NAME Invalid argument: PROFILE_NAME
	80	APPC_ERR_UNEXPECTED_ERROR Unexpected error encountered.
	90	APPC_ERR_UNEXPECTED_DOS_ERROR Unexpected OS/2 error encountered.
	100	APPC_ERROR Call APPCGetSystemError() to get error information returned in PRIMARY_ERROR_CODE and SECONDARY_ERROR_CODE.

Example: response to start
 call APPCAcceptStart
 (ConvID, # Handle is returned
 AcceptProfileName, # Profile name = TP name
 State_Switch) # APPC_Automatic

19.5.2 APPCConvertA2E()

The **APPCConvertA2E()** subroutine is called to perform ASCII to EBCDIC conversion. Note that the conversion is performed in place, meaning that the contents of the ASCII string variable will be replaced with the converted EBCDIC string.

APPCConvertA2E() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCConvertA2E( integer:CONVERSATION_ID,  
                           string:STRING,  
                           integer:INDEX_START,  
                           integer:INDEX_END,  
                           integer:ASCII_TABLE,  
                           integer:EBCDIC_TABLE )  
  
    library "eslappc"
```

19.5.2.1 Arguments

CONVERSATION_ID is an integer variable containing the conversation identifier, or handle, for the APPC conversation returned by either **APPCStart()** or **APPCAcceptStart()**.

STRING is a string variable containing the ASCII string to be converted to EBCDIC. When conversion is complete, this string will contain EBCDIC characters.

INDEX_START is an integer variable indicating the position of the first character in the string to be converted. The first character of a string is indexed by one. Zero can be used as a default value to mean start with the first character.

INDEX_END is an integer variable indicating the position of the last character in the string to be converted. The number of characters in the string is equal to the index to the last character position. Zero can be used as a default value to indicate the last character.

ASCII_TABLE is an integer variable representing the ASCII code page to use for conversion. If this value is zero, the ASCII code page specified in the EASEL OS/2 EE APPC profile is used.

EBCDIC_TABLE is an integer variable representing the EBCDIC code page to use for conversion. If this value is zero, the EBCDIC code page specified in the EASEL OS/2 EE APPC profile is used.

19.5.2.2 Call

A call to **APPCConvertA2E()** is made according to the following syntax:

Model

```
call APPCConvertA2E( CONVERSATION_ID,  
                     STRING,  
                     INDEX_START,  
                     INDEX_END,  
                     ASCII_TABLE,  
                     EBCDIC_TABLE )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	18	APPC_ERR_CONVERSION_ERROR Error on character conversion.
	21	APPC_ERR_INV_CONV_ID Invalid argument: CONVERSATION_ID
	28	APPC_ERR_INV_INDEX_START Invalid argument: INDEX_START
	29	APPC_ERR_INV_INDEX_END Invalid argument: INDEX_END
	31	APPC_ERR_INV_STRING_VAR Invalid string variable.
	40	APPC_ERR_INV_ASCII_TABLE Invalid argument: ASCII_TABLE
	41	APPC_ERR_INV_EBCDIC_TABLE Invalid argument: EBCDIC_TABLE
	70	APPC_ERR_UNEXPECTED_EASEL_ERROR Unexpected error on call to EASEL OS/2 EE routine.
	80	APPC_ERR_UNEXPECTED_ERROR Unexpected error encountered.
	90	APPC_ERR_UNEXPECTED_DOS_ERROR Unexpected OS/2 error encountered.

Example: copy 0 to Zero
 call APPCConvertA2E
 (ConvID, # Handle from Start/AcceptStart
 String, # Convert string to EBCDIC
 Zero, Zero, # Convert the entire string
 Zero, Zero) # Use the default code pages

19.5.3 APPCConvertE2A()

The **APPCConvertE2A()** subroutine is called to perform EBCDIC to ASCII conversion. The conversion is performed in place, meaning that the EBCDIC contents of the string variable will be replaced with the converted ASCII string.

Note: This subroutine would not need to be used if a value of **APPC_Convert** is used for the **AUTO_CONVERT** argument on the **APPCSendString()** call.

APPCConvertE2A() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

Declaration

```
subroutine APPCConvertE2A( integer:CONVERSATION_ID,  
                           string:STRING,  
                           integer:INDEX_START,  
                           integer:INDEX_END,  
                           integer:ASCII_TABLE,  
                           integer:EBCDIC_TABLE )  
  
    library "eslappc"
```

19.5.3.1 Arguments

CONVERSATION_ID is an integer variable containing the conversation identifier, or handle, for the APPC conversation returned by either **APPCStart()** or **APPCAcceptStart()**.

STRING is a string variable containing the EBCDIC string to be converted to ASCII. When conversion is complete, this string will contain ASCII characters.

INDEX_START is an integer variable indicating the position of the first character in the string to be converted. The first character of a string is indexed by one. Zero can be used as a default value to mean start with the first character.

INDEX_END is an integer variable indicating the position of the last character in the string to be converted. The number of characters in the string is equal to the index to the last character position. Zero can be used as a default value to indicate the last character.

ASCII_TABLE is an integer variable representing the ASCII code page to use for conversion. If this value is zero, the ASCII code page specified in the EASEL OS/2 EE APPC profile is used.

EBCDIC_TABLE is an integer variable representing the EBCDIC code page to use for conversion. If this value is zero, the EBCDIC code page specified in the EASEL OS/2 EE APPC profile is used.

19.5.3.2 Call

A call to **APPCConvertE2A()** is made according to the following syntax:

Model

```
call APPCConvertE2A( CONVERSATION_ID,  
                     STRING,  
                     INDEX_START,  
                     INDEX_END,  
                     ASCII_TABLE,  
                     EBCDIC_TABLE )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	18	APPC_ERR_CONVERSION_ERROR Error on character conversion.
	21	APPC_ERR_INV_CONV_ID Invalid argument: CONVERSATION_ID
	28	APPC_ERR_INV_INDEX_START Invalid argument: INDEX_START
	29	APPC_ERR_INV_INDEX_END Invalid argument: INDEX_END
	31	APPC_ERR_INV_STRING_VAR Invalid string variable.
	40	APPC_ERR_INV_ASCII_TABLE Invalid argument: ASCII_TABLE
	41	APPC_ERR_INV_EBCDIC_TABLE Invalid argument: EBCDIC_TABLE
	70	APPC_ERR_UNEXPECTED_EASEL_ERROR Unexpected error on call to EASEL OS/2 EE routine.
	80	APPC_ERR_UNEXPECTED_ERROR Unexpected error encountered.
	90	APPC_ERR_UNEXPECTED_DOS_ERROR Unexpected OS/2 error encountered.

Example: copy 0 to Zero
 call APPCConvertE2A
 (ConvID, # Handle from Start/AcceptStart
 String, # Convert string to ASCII
 Zero, Zero, # Convert the entire string
 Zero, Zero) # Use the default code pages

19.5.4 APPCEnd()

The **APPCEnd()** subroutine is called to terminate an existing conversation with a partner program on another machine.

APPCEnd() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCEnd(integer:CONVERSATION_ID,  
                  integer:SYNC_LEVEL)  
    library "eslappc"
```

19.5.4.1 Arguments

CONVERSATION_ID is an integer variable containing the conversation identifier, or handle, for the APPC conversation returned by either **APPCStart()** or **APPCAcceptStart()**.

SYNC_LEVEL is an integer variable used to coordinate ending a conversation with the partner. Valid constants are:

APPC_Aband	Abnormal termination. If your program was sending data, forces the data in the buffer to be sent prior to deallocating the conversation.
APPC_Flush	(Default) Forces the data to be sent before ending the conversation.
APPC_Confirm	Requests confirmation and deallocates the conversation following receipt of an APPC_Confirmed event notification. The conversation is NOT deallocated if an APPC_ReceiveError or APPC_SendError event notification is received.
APPC_SyncLevel	This defaults to APPC_Confirm if the conversation synchronization level is APPC_Confirm or to APPC_Flush if the conversation synchronization level is APPC_None . The conversation synchronization level is specified as a argument on the APPCStart() subroutine call.

19.5.4.2 Call

A call to **APPCEnd()** is made according to the following syntax:

Model

```
call APPCEnd( CONVERSATION_ID,  
              SYNC_LEVEL )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	21	APPC_ERR_INV_CONV_ID Invalid argument: CONVERSATION_ID
	22	APPC_ERR_INV_SYNC_LEVEL Invalid argument: SYNC_LEVEL
	80	APPC_ERR_UNEXPECTED_ERROR Unexpected error encountered.
	90	APPC_ERR_UNEXPECTED_DOS_ERROR Unexpected OS/2 error encountered.

Example:

```

response to stimulus APPC_EVENT APPC_Ended ConvID
call APPCEnd
    ( ConvID, # Handle from Start/AcceptStart
      Sync ) # Use the conversation sync
copy 0 to ConvID # Terminate

# My program is ending
response to MainWindow on close
if (ConvID != 0) then
    call APPCEnd
    ( ConvID, # Handle from Start/AcceptStart
      Sync )
end if

```

19.5.5 APPCFlush()

The **APPCFlush()** subroutine is called to force all buffered information to be sent to the partner program on another machine. **APPCSendString()** buffers information when the default settings are used.

Note: This subroutine would not normally be needed if you are using the automatic interface, because the conversation state is normally changed automatically. An example of when this subroutine would be needed is to force remaining buffered data to be sent in a situation where the last call to **APPCSendString()** did NOT have a **SEND_IND** parameter with a value of **APPC_Flush**.

APPCFlush() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCFlush( integer:CONVERSATION_ID )  
    library "eslappc"
```

19.5.5.1 Arguments

CONVERSATION_ID is an integer variable containing the conversation identifier, or handle, for the APPC conversation returned by either **APPCStart()** or **APPCAcceptStart()**.

19.5.5.2 Call

A call to **APPCFlush()** is made according to the following syntax:

Model

```
call APPCFlush( CONVERSATION_ID )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	21	APPC_ERR_INV_CONV_ID Invalid argument: CONVERSATION_ID
	80	APPC_ERR_UNEXPECTED_ERROR Unexpected error encountered.
	90	APPC_ERR_UNEXPECTED_DOS_ERROR Unexpected OS/2 error encountered.

Example:

```

call APPCSendString
( ConVID,      # Handle from Start/AcceptStart
  String1,     # First data
  NoConvert,   # Do not convert string
  None,        # Add Send to buffer
  More)        # More to send

call APPCSendString
( ConVID,      # Handle from Start/AcceptStart
  String2,     # Second data
  NoConvert,   # Do not convert string
  None,        # Add Send to buffer
  More)        # More to send

call APPCFlush
( ConVID )     # Send all data in buffer

```

19.5.6 APPCGetAcceptProfile()

The **APPCGetAcceptProfile()** subroutine is called to obtain a copy of the profile information previously saved. This call should be used by a program that calls **APPCAcceptStart()**. This routine may be used to verify if an EASEL OS/2 EE APPC profile exists or to display the current profile settings to a user who may update the values. If a profile does not exist, as indicated by a non-zero errorlevel value, the program must call **APPCSetAcceptProfile()** to set the values.

APPCGetAcceptProfile() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCGetAcceptProfile ( string:PROFILE_NAME,  
                                string:INVOCATION_CMD,  
                                integer:INACTIVITY_TIMEOUT,  
                                integer:BUFFER_SIZE,  
                                integer:ASCII_TABLE,  
                                integer:EBCDIC_TABLE )  
  
    library "eslappc"
```

19.5.6.1 Arguments

PROFILE_NAME is a string variable that specifies the name of the profile to be retrieved.

Note: The **PROFILE_NAME** argument is the only one that you need to specify for this subroutine call. If the subroutine executes successfully, all other arguments will be filled in for you with the stored profile information that you are retrieving. The other arguments are described below only to help you evaluate their contents.

INVOCATION_CMD is a string variable that is set to the command issued by the partner TP (as a result of a call to **APPCAcceptStart()**). This argument is specified when your EASEL OS/2 EE program is started by its partner dynamically. In this case the command name includes a complete file name and can include a fully-qualified path where appropriate, as shown in the following examples:

```
eslrun.exe myeslpgm  
eslrun.exe d:\mypgms\myeslpgm  
c:\easel-ee\eslrun.exe chesspgm -d EASEL OS/2 EEInvocationParm black
```

INACTIVITY_TIMEOUT is an integer variable that is set to the timer value, in seconds, allowing state switching to occur automatically when a conversation with authority to send is inactive for more than the specified number of seconds. This is a value that may require tuning; five seconds is a reasonable initial value. If zero, no check is made to see if the partner application wants to send data between sends. *This prevents automatic switching from being initiated by the partner!* If automatic switching is disabled, your program, when it has completed sending, must be sure to call the

APPCSendString() subroutine with the **MORE_TO_SEND** argument set to **APPC_ReadyToReceive** or else call the **APPCReadyToReceive()** subroutine.

BUFFER_SIZE is an integer between 0 and 65500 (default: 2048) that is set to the buffer size, in bytes, to be used to send and receive data. If zero, a default buffer size of 2K bytes is used. The buffer size must be large enough to contain the largest number of characters or bytes of data to be sent in one call.

ASCII_TABLE is an integer value that is set to the ASCII code page used for conversions. If zero, the current system code page is used.

EBCDIC_TABLE is an integer value that is set to the EBCDIC code page used for conversions. If zero, the international code page, 500, is used.

19.5.6.2 Call

A call to **APPCGetAcceptProfile()** is made according to the following syntax:

Model

```
call APPCGetAcceptProfile( PROFILE_NAME,
                           INVOCATION_CMD,
                           INACTIVITY_TIMEOUT,
                           BUFFER_SIZE,
                           ASCII_TABLE,
                           EBCDIC_TABLE )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	8	APPC_ERR_ON_FILE_OPEN Error opening file or file does not exist.
	9	APPC_ERR_ON_FILE_READ Error reading file or profile not in profile file.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	32	APPC_ERR_INV_PROFILE_NAME Invalid argument: PROFILE_NAME

Example: response to start
 call APPCGetAcceptProfile
 (Profile,
 Invoke_CMD,
 Timeout,
 Buffer_Size,
 ASCII_Table,
 EBCDIC_Table)

19.5.7 APPCGetAcceptProfileNames()

The **APPCGetAcceptProfileNames()** subroutine is called to return the list of profile names currently stored in the EASEL OS/2 EE APPC profile file (ESLAPPCA.PRO) used with EASEL OS/2 EE APPC acceptor programs. One use for this subroutine would be to retrieve the profile names and then display them to the user so that the user could select a profile either to update or to use in an EASEL OS/2 EE APPC application. When you are updating acceptor profile information, this subroutine should be used in combination with the **APPCGetAcceptProfile()** and **APPCSetAcceptProfile()** subroutines.

APPCGetAcceptProfileNames() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCGetAcceptProfileNames( string:PROFILE_NAMES )  
    library eslappc
```

19.5.7.1 Arguments

PROFILE_NAMES is a string variable that receives all of the profile names currently in the acceptor profile file (ESLAPPCA.PRO). The names are returned delimited by forward slashes (/).

19.5.7.2 Call

A call to **APPCGetAcceptProfileNames()** is made according to the following syntax:

Model

```
call APPCGetAcceptProfileNames( PROFILE_NAMES )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	8	APPC_ERR_ON_FILE_OPEN Error opening file.
	9	APPC_ERR_ON_FILE_READ Error reading file.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	31	APPC_ERR_INV_STRING_VAR Invalid string variable.

Unexpected error on call to EASEL
OS/2 EE routine.

Example: response to Edit_Initiator_Profiles
 call APPCGetAcceptProfileNames
 (ProfileNames)

19.5.8 APPCGetDLLData()

The **APPCGetDLLData()** subroutine is called from a user-written DLL subroutine to obtain a buffer of data sent by a partner application upon notification that the data has become available.

The declarations for this subroutine and for the integer constants that can be used with it are found in the EASEL OS/2 EE APPC "C" language include file, ESLAPPC.H.

APPCGetDLLData() is declared in the include file, ESLAPPC.H., according to the following syntax:

"C" Language Declaration

```
USHORT EXPENTRY APPCGetDLLData( ULONG CONVERSATION_ID,  
                                PPBYTE BUFFER_POINTER,  
                                PUSHORT BUFFER_LENGTH );
```

19.5.8.1 Arguments

CONVERSATION_ID is an unsigned long value containing the conversation identifier, or handle, for the APPC conversation returned by either **APPCStart()** or **APPCAcceptStart()**.

BUFFER_POINTER is a pointer to a pointer variable that is set to the address of a buffer. This buffer contains the data received from the partner. The pointer to the buffer is returned. The buffer must be freed in the following manner:

```
DosFreeSeg ( SELECTOROF(*BUFFER_POINTER));
```

BUFFER_LENGTH is a pointer to an unsigned short value which will contain the length of the data buffer.

19.5.8.2 Call

A call to **APPCGetDLLData()** is made according to the following syntax:

Model

```
APPCGetDLLData( CONVERSATION_ID,  
                BUFFER_POINTER,  
                BUFFER_LENGTH );
```

<i>Returns:</i>	Value	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	21	APPC_ERR_INV_CONV_ID Invalid argument: CONVERSATION_ID
	44	APPC_ERR_INV_BUFFER_POINTER Invalid argument: BUFFER_POINTER
	45	APPC_ERR_INV_BUFFER_LENGTH Invalid argument: BUFFER_LENGTH Pointer is zero.
	80	APPC_ERR_UNEXPECTED_ERROR Unexpected error encountered.
	90	APPC_ERR_UNEXPECTED_DOS_ERROR Unexpected OS/2 error encountered.

Example: In the EASEL OS/2 EE program:

```
# Response to data from partner
response to stimulus APPC_EVENT APPC_Data ConvID
call MyAppGetData
    ( ConvID, # Handle from Start/AcceptStart
      Int1,   # From BufInt1
      String1, # From BufStr1
      String2, # From BufStr2
      String3) # From BufStr3
```

In the user-written DLL routine:

```
/* MyAppGetData, a routine written in C */

#include <eslappc.h> /* Defines to call functions */

typedef struct { /* Format of data to send/receive */
    unsigned long BufInt1;
    char          BufStr1[80];
    char          BufStr2[16];
    char          BufStr3[40];
} INPUTAREA;

INPUTAREA far *MyBuf;
PBYTE      pData;
```

```

unsigned long   ConvID;
unsigned short  Length;

APPCGetDLLData
    ( ConvID,      /* Handle from Start/AcceptStart */
      &pData,      /* Buffer returned */
      &Length);   /* Length of data in buffer */

MyBuf = (INPUTAREA far *)pData;
// Process data received
:
DosFreeSeg (SELECTOROF(pData));

```

19.5.9 APPCGetInfo()

The **APPCGetInfo()** subroutine is called to obtain a copy of the conversation characteristics for the CONVERSATION_ID specified in the call.

This subroutine is provided as an aid to the EASEL OS/2 EE programmer. It returns information to help debug configuration problems or to follow traces. Many of the arguments are identical to ones used in the OS/2 Extended Edition Communications Manager configuration profiles. It should be called after an **APPC_Started** event notification is received.

APPCGetInfo() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCGetInfo( integer:CONVERSATION_ID,  
                        string:APPC_TP_ID,  
                        integer:APPC_CONVERSATION_ID,  
                        string:NET_NAME,  
                        string:LOCAL_LU_NAME,  
                        string:LOCAL_LU_ALIAS_NAME,  
                        string:PARTNER_LU_ALIAS_NAME,  
                        string:PARTNER_LU_UNINTERPRETED_NAME,  
                        string:FULLYQUAL_PARTNER_LU_NAME,  
                        string:PARTNER_TP_NAME,  
                        string:MODE_NAME,  
                        integer:SYNC_LEVEL,  
                        string:USER_ID )  
  
    library "eslappc"
```

19.5.9.1 Arguments

CONVERSATION_ID is an integer variable containing the EASEL OS/2 EE APPC identifier for the conversation. This identifier is obtained when the user issues a call to the **APPCStart()** or **APPCAcceptStart()** subroutines and is associated with an EASEL OS/2 EE APPC conversation. It should not be confused with the APPC_CONVERSATION_ID argument described below.

Note: The CONVERSATION_ID argument is the only one that you need to specify for this subroutine call. If the subroutine executes successfully, all other arguments will be filled in for you with the conversation information that you are retrieving. The other arguments are described below only to help you evaluate their contents.

APPC_TP_ID is a string variable that is set to the identifier of the transaction program. This string represents APPC_TP_ID as a sequence of hexadecimal digits. This is the value returned when a **TP_STARTED** or a **RECEIVE_ALLOCATE** APPC verb is issued.

APPC_CONVERSATION_ID is an integer variable that is set to the APPC identifier of the conversation. This is the value returned when an **MC_ALLOCATE** or a **RECEIVE_ALLOCATE** APPC verb is issued.

NET_NAME is a string variable that is set to the name of the network in which the logical unit (LU) of the local transaction program resides. This is the network name configured in the SNA base profile. This EBCDIC value is returned as an ASCII string.

LOCAL_LU_NAME is a string variable that is set to the name of the logical unit (LU) of the local transaction program. This is the LU name configured in the corresponding Logical Unit Profile. This EBCDIC value is returned as an ASCII string.

LOCAL_LU_ALIAS_NAME is a string variable that is set to the locally-known name for the logical unit (LU). This name was established during configuration of OS/2 Extended Edition Communications Manager on the *Create/Change Local APPC Logical Unit Profile* panel. It is also identified in the Partner LU Profile.

PARTNER_LU_ALIAS_NAME is a string variable that is set to the locally-known name of the Partner Logical Unit Profile that was established during configuration.

PARTNER_LU_UNINTERPRETED_NAME is a string variable that is set to the name of the partner LU. This is the name sent to the System Service Control Point while attempting to activate a session with the partner. This EBCDIC value is returned as an ASCII string.

FULLYQUAL_PARTNER_LU_NAME is a string variable that is set to the fully-qualified network name of the remote logical unit (LU) that is defined as a partner for the local LU. The fully-qualified name consists of the name of the network of the partner LU, a period, and the LU name as it is known in the network of the partner LU. This EBCDIC value is returned as an ASCII string.

PARTNER_TP_NAME is a string variable that is set to the name of the partner transaction program (TP). If the partner is using OS/2 Extended Edition Communications Manager, this is the TP name specified in the partner's configuration using the Remotely Attachable Transaction Program Profile. Its maximum length is 64 characters. This EBCDIC value is returned as an ASCII string.

MODE_NAME is a string variable that is set to the mode name. The mode identifies the characteristics of the allocated session. It is identified in the Partner LU Profile during configuration. This EBCDIC value is returned as an ASCII string.

SYNC_LEVEL is an integer variable that is set to the synchronization level that the local and partner TP's can use. Its values are:

- APPC_None**

The conversation cannot use any EASEL OS/2 EE APPC subroutines related to synchronization and will not recognize any partners that use them.
- APPC_Confirm**

The conversation can use EASEL OS/2 EE APPC subroutines related to synchronization and must not ignore a partner that uses them.

USER_ID is a string-variable that is set to the user ID that the initiating program provided in the incoming allocation request. This EBCDIC value is returned as an ASCII string.

19.5.9.2 Call

A call to **APPCGetInfo()** is made according to the following syntax:

Model

```
call APPCGetInfo( CONVERSATION_ID,  
                  APPC_TP_ID,  
                  APPC_CONVERSATION_ID,  
                  NET_NAME,  
                  LOCAL_LU_NAME,  
                  LOCAL_LU_ALIAS_NAME,  
                  PARTNER_LU_ALIAS_NAME,  
                  PARTNER_LU_UNINTERPRETED_NAME,  
                  FULLYQUAL_PARTNER_LU_NAME,  
                  PARTNER_TP_NAME,  
                  MODE_NAME,  
                  SYNC_LEVEL,  
                  USER_ID )
```

Returns:	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	5	APPC_ERR_INFO_NOT_AVAILABLE Information not available until APPC_Started event notification is received.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.

21	APPC_ERR_INV_CONV_ID Invalid argument: CONVERSATION_ID
70	APPC_ERR_UNEXPECTED_EASEL_ERROR Unexpected error on call to EASEL OS/2 EE routine.
80	APPC_ERR_UNEXPECTED_ERROR Unexpected error encountered.
90	APPC_ERR_UNEXPECTED_DOS_ERROR Unexpected OS/2 error encountered.
100	APPC_ERROR Call APPCGetSystemError to get error information returned in PRIMARY_ERROR_CODE and SECONDARY_ERROR_CODE.

Example: response to start
 call APPCGetInfo
 (Conv_ID, # ID returned from APPCStart
 APPC_TP_ID, #
 APPC_Conv_ID, #
 NetWorkName, #
 LocalLU, #
 LocalLU_Alias, # "LU1" example name
 PartnerLU_Alias, # "LU4" example name
 PartnerLU_UninterpretedName, #
 PartnerLU_Name, #
 PartnerTP, # "MYPROFILE" example name
 Mode, # "TKNRING" example name
 SyncLevel, # APPC_Confirm example result
 UserID) #

19.5.10 APPCGetInitProfile()

The **APPCGetInitProfile()** subroutine is called to obtain a copy of the profile information previously saved. This call should be used by a program that calls **APPCStart()**. This routine may be used to verify if a profile exists or to display the current profile settings to a user who may update the values. If a profile does not exist, as indicated by a non-zero errorlevel value, the program must call **APPCSetInitProfile()** to set the values.

APPCGetInitProfile() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCGetInitProfile( string:PROFILE_NAME,  
                              string:LOCAL_LU_ALIAS_NAME,  
                              string:PARTNER_LU_ALIAS_NAME,  
                              string:PARTNER_TP_NAME,  
                              string:MODE_NAME,  
                              string:SECURITY_PARAMETERS,  
                              integer:INACTIVITY_TIMEOUT,  
                              integer:BUFFER_SIZE,  
                              integer:ASCII_TABLE,  
                              integer:EBCDIC_TABLE )  
  
library "eslappc"
```

19.5.10.1 Arguments

PROFILE_NAME is a string variable that specifies the name of the profile to be retrieved.

Note: The PROFILE_NAME argument is the only one that you need to specify for this subroutine call. If the subroutine executes successfully, all other arguments will be filled in for you with the stored profile information that you are retrieving. The other arguments are described below only to help you evaluate their contents.

LOCAL_LU_ALIAS_NAME is a string variable that is set to locally-known name for the logical unit (LU). This name was established during configuration of OS/2 Extended Edition Communications Manager on the *Create/Change Local APPC Logical Unit Profile* panel. Its maximum length is 8 characters. It also must be identified in the Partner LU Profile.

PARTNER_LU_ALIAS_NAME is a string variable that is set to the name of a partner logical unit (LU) profile that was established during configuration. Its maximum length is 8 characters.

PARTNER_TP_NAME is a string variable that is set to the name of the partner transaction program (TP). If the partner is using OS/2 Extended Edition Communications Manager, this is the TP name specified in the partner's configuration using the Remotely Attachable Transaction Program Profile. Its maximum length is 64 characters. If the partner is invoking an EASEL

OS/2 EE program, the partner TP name must be the same as the profile name specified in the **APPCAcceptStart()** subroutine call and the partner TP filename must be ESLAPPCA.EXE.

MODE_NAME is a string variable that is set to the mode name. The mode identifies the characteristics of the allocated session. It is identified in the Partner LU Profile during configuration. Its maximum length is 8 characters.

SECURITY_PARAMETERS is a string variable that is set to the parameters used to validate access to a partner transaction program and its resources. The following values are valid:

none	The conversation does not use security.
same	The user ID is verified locally.
pgm/USERID/PASSWORD	The partner logical unit (LU) validates access with the security information provided. Parameters are separated with a forward slash (/). USERID is the ID of the local user; its maximum length is 10 characters. PASSWORD is the password of the local user; its maximum length is 10 characters.

INACTIVITY_TIMEOUT is an integer variable that is set to the timer value, in seconds, allowing state switching to occur automatically when a conversation with authority to send is inactive for more than the specified number of seconds. This is a value that may require tuning; five seconds is a reasonable initial value. If zero, no check is made to see if the partner application wants to send data between sends. *This prevents automatic switching from being initiated by the partner!* If automatic switching is disabled, your program, when it has completed sending, must be sure to call the **APPCSendString()** subroutine with the MORE_TO_SEND argument set to **APPC_ReadyToReceive** or else call the **APPCReadyToReceive()** subroutine.

BUFFER_SIZE is an integer between 0 and 65500 (default: 2048) that specifies the buffer size, in bytes, to be used to send and receive data. If zero, a default buffer size of 2K bytes is used. The buffer size must be 'arge enough to contain the largest number of characters or bytes of data to be sent in one call.

ASCII_TABLE is an integer value that is set to the ASCII code page used for conversions. If zero, the current system code page is used.

EBCDIC_TABLE is an integer value that is set to the EBCDIC code page used for conversions. If zero, the international code page, 500, is used.

19.5.10.2 Call

A call to **APPCGetInitProfile()** is made according to the following syntax:

Model

```
call APPCGetInitProfile( PROFILE_NAME,  
                        LOCAL_LU_ALIAS_NAME,  
                        PARTNER_LU_ALIAS_NAME,  
                        PARTNER_TP_NAME,  
                        MODE_NAME,  
                        SECURITY_PARAMETERS,  
                        INACTIVITY_TIMEOUT,  
                        BUFFER_SIZE,  
                        ASCII_TABLE,  
                        EBCDIC_TABLE )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	8	APPC_ERR_ON_FILE_OPEN Error opening file or file does not exist.
	9	APPC_ERR_ON_FILE_READ Error reading file or profile not in profile file.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	32	APPC_ERR_INV_PROFILE_NAME Invalid argument: PROFILE_NAME

Example: response to start
 call APPCGetInitProfile
 (Profile,
 MyLU,
 PartnerLU,
 ReceiveTPName,
 ModeName,
 NoSecurity,
 Timeout,
 Buffer_Size,
 ASCII_Table,
 EBCDIC_Table)

19.5.11 APPCGetInitProfileNames()

The **APPCGetInitProfileNames()** subroutine is called to return the list of profile names currently stored in the EASEL OS/2 EE APPC profile file (ESLAPPCI.PRO) used with EASEL OS/2 EE APPC initiator programs. One use for this subroutine would be to retrieve the profile names and then display them to the user so that the user could select a profile either to update or to use in an EASEL OS/2 EE APPC application. When you are updating initiator profile information, this subroutine should be used in combination with the **APPCGetInitProfile()** and **APPCSetInitProfile()** subroutines.

APPCGetInitProfileNames() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCGetInitProfileNames( string:PROFILE_NAMES )  
    library "eslappc"
```

19.5.11.1 Arguments

PROFILE_NAMES is a string variable that receives all of the profile names currently in the initiator profile file (ESLAPPCI.PRO). The names are returned delimited by forward slashes (/).

19.5.11.2 Call

A call to **APPCGetInitProfileNames()** is made according to the following syntax:

Model

```
call APPCGetInitProfileNames( PROFILE_NAMES )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	8	APPC_ERR_ON_FILE_OPEN Error opening file.
	9	APPC_ERR_ON_FILE_READ Error reading file.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	31	APPC_ERR_INV_STRING_VAR Invalid string variable.

Unexpected error on call to EASEL
OS/2 EE routine.

Example: response to Edit_Initiator_Profiles
call APPCGetInitProfileNames
(ProfileNames)

19.5.12 APPCGetString()

The **APPCGetString()** subroutine is called to obtain data from a partner application when notified that the data has become available.

APPCGetString() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCGetString( integer:CONVERSATION_ID,  
                        string:STRING_VAR,  
                        integer:AUTO_CONVERT )  
  
    library "eslappc"
```

19.5.12.1 Arguments

CONVERSATION_ID is an integer variable containing the conversation identifier, or handle, of the particular conversation returned by either **APPCStart()** or **APPCAcceptStart()**.

STRING_VAR is a string variable name whose value is set to the data received from the partner.

AUTO_CONVERT is an integer variable which indicates whether automatic conversion from EBCDIC to ASCII of the contents of STRING_VAR should be performed. This conversion uses the code page tables specified in the EASEL OS/2 EE APPC profile. Data from a host computer may be in EBCDIC, data on the workstation is in ASCII. Valid constants are:

- APPC_NoConvert** (Default) Requests that conversion not be done.
- APPC_Convert** Requests that the string be converted from EBCDIC to ASCII.

19.5.12.2 Call

A call to **APPCGetString()** is made according to the following syntax:

Model

```
call APPCGetString( CONVERSATION_ID,  
                   STRING_VAR,  
                   AUTO_CONVERT )
```

Returns:	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.

16	APPC_ERR_OUT_OF_MEMORY Out of memory.
18	APPC_ERR_CONVERSION_ERROR Error on character conversion
21	APPC_ERR_INV_CONV_ID Invalid argument: CONVERSATION_ID
24	APPC_ERR_INV_AUTO_CONVERT Invalid argument: AUTO_CONVERT
31	APPC_ERR_INV_STRING_VAR Invalid string variable.
51	APPC_ERR_NO_DATA_AVAILABLE APPCGetString called when no data was received.
52	APPC_ERR_SYSTEM_ERROR_PENDING APPCGetString called when system error was pending.
70	APPC_ERR_UNEXPECTED_EASEL_ERROR Unexpected error on call to EASEL OS/2 EE routine.
80	APPC_ERR_UNEXPECTED_ERROR Unexpected error encountered.
90	APPC_ERR_UNEXPECTED_DOS_ERROR Unexpected OS/2 error encountered.

Example:

```

response to stimulus APPC_EVENT APPC_Data ConvID
  call APPCGetString
    ( ConvID,      # Handle from Start/AcceptStart
      GetMove,    # String to hold partner's move
      NoConvert )  # Do not convert data

  call APPCSendString
    ( ConvID,      # Handle from Start/AcceptStart
      ResponseMove, # String containing next move
      NoConvert,   # Do not convert data
      Confirm,     # Confirm receipt
      RdytoRecv )  # This send expects reply

# Response to confirm request from partner
response to stimulus APPC_EVENT APPC_ConfirmRequest ConvID
  call APPCSendConfirmed
    ( ConvID )      # Handle from Start/AcceptStart

```

19.5.13 APPCGetSystemError()

The **APPCGetSystemError()** subroutine is called to get error information when a system error is signalled; it should be called in a response to the **APPC_SystemError** event notification. Since much of the communications processing occurs asynchronously to the EASEL OS/2 EE programs, errors may occur after a subroutine call. System errors may be flagged by EASEL OS/2 EE APPC or by an APPC service error. A system error is reported only after error recovery has failed. This routine is provided to return detailed information so that the user or a communications expert can diagnose the problem. Your EASEL OS/2 EE program should display or log the error.

Note: Do not call **APPCGetSystemError()** for errors signalled by a partner application.

APPCGetSystemError() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCGetSystemError( integer:CONVERSATION_ID,  
                              integer:WORK_REQUEST_ERROR,  
                              integer:WORK_REQUEST_IDENTIFIER,  
                              integer:APPC_VERB,  
                              integer:PRIMARY_ERROR_CODE,  
                              integer:SECONDARY_ERROR_CODE )  
  
    library "eslappc"
```

19.5.13.1 Arguments

CONVERSATION_ID is an integer variable containing the conversation identifier, or handle, for the APPC conversation returned by either **APPCStart()** or **APPCAcceptStart()**.

WORK_REQUEST_ERROR is an integer variable that receives a number representing the error condition encountered by the work request. The error number can be interrogated using the integer constants listed below.

Every call to an EASEL OS/2 EE APPC subroutine generates one or more low-level *work requests* internally to EASEL OS/2 EE APPC support, representing discrete APPC tasks, for example "send data" or "receive buffer." The work requests are queued when the subroutine is called, and then serviced when conditions are appropriate for their processing.

APPC_ERR_OUT_OF_MEMORY

Out of memory.

APPC_ERR_INV_TO_ISSUE_CNFRM_REQ

Invalid to call **APPCRequestConfirm()** subroutine or use **APPC_Confirm** parameter value in **APPCSendString()**, **APPCSendDLLData()**, or **APPCend()** subroutines when not using **APPC_Confirm** value for **SYNC_LEVEL** on **APPCStart()**.

APPC_ERR_INV_TO_ISSUE_CONFIRMED

Invalid to call **APPDSendConfirmed()** subroutine when **APPC_ConfirmRequest** event notification not received.

APPC_ERR_CONFIRM_REQ_IGNORED

APPC_ConfirmRequest event notification ignored because conversation not in *Send* state.

APPC_ERR_INV_TO_ISSUE_RECV_NTFY

Invalid to call **APPCReceiveNotify()** subroutine in automatic interface.

APPC_ERR_NO_REMOTE_TP_PARAM

The Communications Manager configuration for remotely attachable TPs did not specify the EASEL OS/2 EE APPC acceptor profile name in the TP start-up parameters field.

APPC_ERR_BASIC_CONV_NOT_VALID

An APPC TP issued an allocate verb that attempted to start an EASEL OS/2 EE APPC acceptor program in "basic" conversation mode. Only "mapped" conversations are supported. Specify Mapped as the value for Conversation type on the Communications Manager *Create/Change Remotely Attachable Transaction Program Profile* panel to eliminate this error. Your EASEL OS/2 EE APPC program will not be started, and the APPC TP will get an APPC error.

APPC_ERR_CONV_ENDING_REQ_PURGED

Work request will not be processed because this APPC conversation has received either a normal deallocation from its partner or a non-recoverable system error has occurred.

APPC_ERR_UNEXPECTED_ERROR

Unexpected error encountered.

APPC_ERR_UNEXPECTED_DOS_ERROR

Unexpected OS/2 error encountered.

APPC_ERROR

Look at values returned in PRIMARY_ERROR_CODE and SECONDARY_ERROR_CODE.

WORK_REQUEST_IDENTIFIER is an integer variable that receives a number representing the EASEL OS/2 EE APPC work request that was executing when the error was encountered. The number can be interrogated using the integer constants listed below.

APPC_AcceptStart
APPC_ConvertString
APPC_GetInfo
APPC_ReceiveBuffer
APPC_ReceiveNotify
APPC_ReceiveString
APPC_RequestConfirm
APPC_RequestRdyToReceive

APPC_RequestSend
APPC_SendAbend
APPC_SendConfirmed
APPC_SendData
APPC_SendErrorItem
APPC_SendFlush
APPC_SendTerminate
APPC_Start
APPC_TestReqSend

APPC_VERB is an integer variable that receives a number representing the APPC verb reporting the error. The number can be interrogated using the integer constants listed below.

AP_M_ALLOCATE
AP_M_CONFIRM
AP_M_CONFIRMED
AP_M_DEALLOCATE
AP_M_FLUSH
AP_M_GET_ATTRIBUTES
AP_M_PREPARE_TO_RECEIVE
AP_M_RECEIVE_AND_POST
AP_M_RECEIVE_AND_WAIT
AP_M_RECEIVE_IMMEDIATE
AP_M_REQUEST_TO_SEND
AP_M_SEND_DATA
AP_M_SEND_ERROR
AP_M_TEST_RTS
AP_RECEIVE_ALLOCATE
AP_TP_ENDED
AP_TP_STARTED
SV_GET_CP_CONVERT_TABLE

PRIMARY_ERROR_CODE is an integer variable that is set to primary return code from the APPC verb.

SECONDARY_ERROR_CODE is an integer variable that is set to secondary return code, if any, associated with the primary code for the APPC verb.

19.5.13.2 Call

A call to **APPCGetSystemError()** is made according to the following syntax:

Model

```
call APPCGetSystemError( CONVERSATION_ID,  
                          WORK_REQUEST_ERROR,  
                          WORK_REQUEST_IDENTIFIER,  
                          APPC_VERB,  
                          PRIMARY_ERROR_CODE,  
                          SECONDARY_ERROR_CODE )
```

Returns:	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	21	APPC_ERR_INV_CONV_ID Invalid argument: CONVERSATION_ID
	59	APPC_ERR_NO_SYSTEM_ERROR_DATA APPCGetSystemError() was called when no system error data was avail- able.
	80	APPC_ERR_UNEXPECTED_ERROR Unexpected error encountered.
	90	APPC_ERR_UNEXPECTED_DOS_ERROR Unexpected OS/2 error encountered.

Example: response to stimulus APPC_EVENT APPC_SystemError ConvID

```
call APPCGetSystemError  
  ( ConvID,                                    # ID from Start/AcceptStar  
    ErrorEncountered,                        # Error number  
    WorkRequestNumber,                       # Work request in process  
    APPCLastVerb,                            # APPC Verb reporting error  
    PrimaryReturnCode,                       # integer (6 digit number)  
    SecondaryReturnCode )                    # integer (8 digit number)  
if (errorlevel = ?) ...
```

19.5.14 APPCReadyToReceive()

The **APPCReadyToReceive()** subroutine is called to send an indication to the partner program that your program is ready to receive data. This gives the partner program authority to send. See also the **APPC_ReadyToReceive** value of the **MORE_TO_SEND** argument on **APPCSendString()**.

Note: This subroutine would not normally be needed if you are using the automatic interface, since the conversation state is normally changed automatically. An example of when this subroutine would be needed is to force a change in conversation state from *Send* to *Receive* when the last call to **APPCSendString()** had a **MORE_TO_SEND** argument with a value of **APPC_MoreData**.

APPCReadyToReceive() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCReadyToReceive( integer:CONVERSATION_ID )  
    library "eslappc"
```

19.5.14.1 Arguments

CONVERSATION_ID is an integer variable containing the conversation identifier, or handle, for the APPC conversation returned by either **APPCStart()** or **APPCAcceptStart()**.

19.5.14.2 Call

A call to **APPCReadyToReceive()** is made according to the following syntax:

Model

```
call APPCReadyToReceive( CONVERSATION_ID )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	21	APPC_ERR_INV_CONV_ID Invalid argument: CONVERSATION_ID
	80	APPC_ERR_UNEXPECTED_ERROR Unexpected error encountered.
	90	APPC_ERR_UNEXPECTED_DOS_ERROR Unexpected OS/2 error encountered.

Example:

```
# Now In Send State
response to stimulus APPC_EVENT APPC_SendState ConvID
call APPCSendString
    ( ConvID,      # Handle from Start/AcceptStart
      String1,    # String to send
      NoConvert,  # Do not convert string
      Force,      # Send data now
      More )      # Will send more data

call APPCSendString
    ( ConvID,      # Handle from Start/AcceptStart
      String2,    # String to send
      NoConvert,  # Do not convert string
      Force,      # Send data now
      More )      # Will send more data
:
                                # No more data to send
# End of send
call APPCReadyToReceive
    ( ConvID )      # Handle from Start/AcceptStart
```


19.5.15 APPCReceiveNotify()

The **APPCReceiveNotify()** subroutine is called to request receipt of data. When data is available it is indicated by the **APPC_Data** event notification. Call **APPCGetString()** to get the data.

Note: This routine must not be used in the automatic interface, or a system error will be generated.

APPCReceiveNotify() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCReceiveNotify( integer:CONVERSATION_ID )  
    library "eslappc"
```

19.5.15.1 Arguments

CONVERSATION_ID is an integer variable containing the conversation identifier, or handle, for the APPC conversation returned by either **APPCStart()** or **APPCAcceptStart()**.

19.5.15.2 Call

A call to **APPCReceiveNotify()** is made according to the following syntax:

Model

```
call APPCReceiveNotify( CONVERSATION_ID )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	21	APPC_ERR_INV_CONV_ID Invalid argument: CONVERSATION_ID
	80	APPC_ERR_UNEXPECTED_ERROR Unexpected error encountered.
	90	APPC_ERR_UNEXPECTED_DOS_ERROR Unexpected OS/2 error encountered.

Example: call APPCReceiveNotify
 (ConvID) # Handle from Start/AcceptStart

 # Data is available
 response to stimulus APPC_EVENT APPC_Data ConvID
 call APPCGetString
 (ConvID, # Handle from Start/AcceptStart
 String1, # Get data
 NoConvert) # Do not convert string

19.5.16 APPCRequestConfirm()

The **APPCRequestConfirm()** subroutine is called to send a request for confirmation to the partner program to synchronize processing.

This call also flushes all buffered information to the partner program. The **APPCSendString()** subroutine buffers information when the default settings are used.

APPCRequestConfirm() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCRequestConfirm( integer:CONVERSATION_ID )  
    library "eslappc"
```

19.5.16.1 Arguments

CONVERSATION_ID is an integer variable containing the conversation identifier, or handle, for the APPC conversation returned by either **APPCStart()** or **APPCAcceptStart()**.

19.5.16.2 Call

A call to **APPCRequestConfirm()** is made according to the following syntax:

Model

```
call APPCRequestConfirm( CONVERSATION_ID )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	21	APPC_ERR_INV_CONV_ID Invalid argument: CONVERSATION_ID
	80	APPC_ERR_UNEXPECTED_ERROR Unexpected error encountered.
	90	APPC_ERR_UNEXPECTED_DOS_ERROR Unexpected OS/2 error encountered.

Example: call APPCSendString
 (ConvID, # Handle from Start/AcceptStart
 String1, # First data
 NoConvert, # Do not convert string
 None, # Add Send to buffer
 More) # More to send

 call APPCSendString
 (ConvID, # Handle from Start/AcceptStart
 String2, # Second data
 NoConvert, # Do not convert string
 None, # Add Send to buffer
 More) # More to send

 call APPCRequestConfirm
 (ConvID) # Send all data in buffer

19.5.17 APPCRequestToSend()

The **APPCRequestToSend()** subroutine is called to send an indication to the partner program that your program wants to send data. Your program knows that it has been granted permission to send when it receives an **APPC_SendState** event notification.

APPCRequestToSend() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCRequestToSend( integer:CONVERSATION_ID )  
    library "eslappc"
```

19.5.17.1 Arguments

CONVERSATION_ID is an integer variable containing the conversation identifier, or handle, for the APPC conversation returned by either **APPCStart()** or **APPCAcceptStart()**.

19.5.17.2 Call

A call to **APPCRequestToSend()** is made according to the following syntax:

Model

```
call APPCRequestToSend( CONVERSATION_ID )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	21	APPC_ERR_INV_CONV_ID Invalid argument: CONVERSATION_ID
	80	APPC_ERR_UNEXPECTED_ERROR Unexpected error encountered.
	90	APPC_ERR_UNEXPECTED_DOS_ERROR Unexpected OS/2 error encountered.

```

Example:      call APPCGetString
                  ( ConvID,      # Handle from Start/AcceptStart
                    String1,     # Get data
                    NoConvert )  # Do not convert string

      call APPCRequestToSend
        ( ConvID )      # Handle from Start/AcceptStart

      call APPCReceiveNotify
        ( ConvID )      # Handle from Start/AcceptStart

# Now In Send State
response to stimulus APPC_EVENT APPC_SendState ConvID
      call APPCSendString
        ( ConvID,      # Handle from Start/AcceptStart
          String2,     # String to send
          NoConvert,   # Do not convert string
          Force,       # Send data now
          PrepareToReceive ) # This send expects reply

      call APPCReceiveNotify
        ( ConvID )      # Handle from Start/AcceptStart

```

19.5.18 APPCSendConfirmed()

A program calls the **APPCSendConfirmed()** subroutine to notify its partner that it has successfully processed all data received prior to the partner's confirmation request. A program should call this subroutine in a **response to stimulus ... APPC_ConfirmRequest** response definition.

APPCSendConfirmed() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCSendConfirmed( integer:CONVERSATION_ID )
    library "eslappc"
```

19.5.18.1 Arguments

CONVERSATION_ID is an integer variable containing the conversation identifier, or handle, for the APPC conversation returned by either **APPCStart()** or **APPCAcceptStart()**.

19.5.18.2 Call

A call to **APPCSendConfirmed()** is made according to the following syntax:

Model

```
call APPCSendConfirmed( CONVERSATION_ID )
```

Returns:	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	21	APPC_ERR_INV_CONV_ID Invalid argument: CONVERSATION_ID
	80	APPC_ERR_UNEXPECTED_ERROR Unexpected error encountered.
	90	APPC_ERR_UNEXPECTED_DOS_ERROR Unexpected OS/2 error encountered.

Example:

```
# Confirm notification
response to stimulus APPC_EVENT APPC_ConfirmRequest ConVID
if (ProcessInd = 0) then
    call APPCSendConfirmed # Send Confirmed notification
        ( ConVID )        # Handle from Start/AcceptStart
else
    call APPCSendError      # Send Error signal
        ( ConVID,          # Handle from Start/AcceptStart
          BadDataRecv )    # APPC_BadGet
end if
```


19.5.19 APPCSendDLLData()

The **APPCSendDLLData()** subroutine is called from a user-written DLL subroutine to send a buffer of data to a partner. The sender can request that data be sent immediately, can request a confirmation, and can indicate that a reply from the partner is expected before more data will be sent. The buffer must have been previously prepared by the user-written routine.

Note: An **APPC_ConfirmRequest** event is signalled to your EASEL OS/2 EE program when the partner program requests confirmation. Your EASEL OS/2 EE program must respond by either calling **APPCSendConfirmed()** or **APPCSendError()**. If your program calls **APPCSendConfirmed()**, it generates a confirm notification to the partner program. When the partner is an EASEL OS/2 EE program, this triggers its **response to stimulus ... APPC_Confirmed** response statement. If your program calls **APPCSendError()**, it generates an error notification to the partner program. When the partner is an EASEL OS/2 EE program, this triggers **response to stimulus ... APPC_ReceiveError** or **response to stimulus ... APPC_SendError**.

The declarations for this subroutine and for the integer constants that can be used with it are found in the EASEL OS/2 EE APPC "C" language include file, ESLAPPC.H.

APPCSendDLLData() is declared in the include file, ESLAPPC.H, according to the following syntax:

"C" Language Declaration

```
USHORT EXPENTRY APPCSendDLLData( ULONG CONVERSATION_ID,  
                                PBYTE BUFFER_POINTER,  
                                USHORT BUFFER_LENGTH,  
                                USHORT SENDIND,  
                                USHORT MORETOSEND );
```

19.5.19.1 Arguments

CONVERSATION_ID is an unsigned long value containing the conversation identifier, or handle, for the APPC conversation returned by either **APPCStart()** or **APPCAcceptStart()**.

BUFFER_POINTER is a pointer to a buffer of data to send to the partner.

BUFFER_LENGTH is an unsigned short value containing the length of the data buffer.

SENDIND is an unsigned short value used to specify control over when data is actually sent to the partner. Valid constants are:

APPC_None	(Default) Data is accumulated in a buffer to be sent when the buffer is full, or when another request causes the data to be sent.
APPC_Flush	The data is sent immediately.
APPC_Confirm	The data is sent immediately and a confirm notification is requested.

Note: A response to stimulus ...
APPC_ConfirmRequest response definition MUST be included in your EASEL OS/2 EE program if confirmation will be requested by its partner. *Otherwise the conversation will not proceed!* The response to stimulus ... **APPC_Confirmed** response definition is activated when the partner program sends your program a confirmation, in response to your confirmation request.

APPC_SyncLevel	The data is sent based upon the synchronization level of the conversation, established when it was started. If the synchronization level was specified as APPC_None , the data is sent immediately. If specified as APPC_Confirm , the data is sent immediately and a confirm notification is requested.
-----------------------	--

MORETOSEND is an unsigned short value that indicates whether your program has more data to send following the string being sent with this call. Valid constants are:

APPC_MoreData	(Default) Indicates that this application has more data to send before it expects a response.
APPC_ReadyToReceive	Indicates that the application expects to receive data from its partner before it has more data to send.

19.5.19.2 Call

A call to **APPCSendDLLData()** is made according to the following syntax:

Model

```
APPCSendDLLData( CONVERSATION_ID,  
                  BUFFER_POINTER,  
                  BUFFER_LENGTH,  
                  SENDIND,  
                  MORETOSEND );
```

<i>Returns:</i>	Value	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	21	APPC_ERR_INV_CONV_ID Invalid argument: CONVERSATION_ID
	25	APPC_ERR_INV_SEND_IND Invalid argument: SEND_IND
	26	APPC_ERR_INV_MORE_TO_SEND Invalid argument: MORE_TO_SEND
	44	APPC_ERR_INV_BUFFER_POINTER Invalid argument: BUFFER_POINTER
	46	APPC_ERR_LGTH_EXCDS_PRF_BUFSIZE Invalid argument: BUFFER_LENGTH String length exceeds buffer size specified in profile.
	80	APPC_ERR_UNEXPECTED_ERROR Unexpected error encountered.
	90	APPC_ERR_UNEXPECTED_DOS_ERROR Unexpected OS/2 error encountered.

Example: In the EASEL OS/2 EE program:

```
# Response to data from partner
response to stimulus APPC_EVENT APPC_Data ConvID
:
call MyAppSendData
    ( ConvID,      # Handle from Start/AcceptStart
      Int1,        # Fills BufInt1
      String1,     # Fills BufStr1
      String2,     # Fills BufStr2
      String3 )    # Fills BufStr3
```

In the user-written DLL routine:

```
/* MyAppSendData, a routine written in C */

#include <eslappc.h>      /* Defines to call functions */

typedef struct {          /* Format of data to send/receive */
    unsigned long BufInt1;
    char          BufStr1[80];
    char          BufStr2[16];
    char          BufStr3[40];
} INPUTAREA;

INPUTAREA far *MyBuf;

unsigned long ConvID;
unsigned short Length;

APPCSendDLLData
    ( ConvID,          /* Handle from Start/AcceptStart */
      MyBuf,           /* Buffer containing data */
      Length,          /* Length of data in buffer */
      APPC_Confirm,    /* Confirm receipt */
      APPC_ReadyToReceive ); /* Send expects reply */
```

19.5.20 APPCSendError()

The **APPCSendError()** subroutine is called to notify a partner program that an application error has occurred. Use this subroutine in conjunction with **APPCSendConfirmed()** to synchronize processing with the partner program.

APPCSendError() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCSendError( integer:CONVERSATION_ID,  
                        integer:ERROR_ORIGIN )  
    library "eslappc"
```

19.5.20.1 Arguments

CONVERSATION_ID is an integer variable containing the conversation identifier, or handle, for the APPC conversation returned by either **APPCStart()** or **APPCAcceptStart()**.

ERROR_ORIGIN is an integer variable indicating the reason for the error. Valid constants are:

APPC_BadGet	(Default) Specifies that the error was found processing the data just received. This value causes an APPC_ReceiveError event notification to be sent to the partner.
APPC_BadSend.	Specifies that the error occurred when building the data to be sent. This value causes an APPC_SendError event notification to be sent to the partner.

19.5.20.2 Call

A call to **APPCSendError()** is made according to the following syntax:

Model

```
call APPCSendError( CONVERSATION_ID,  
                  ERROR_ORIGIN )
```

Returns:	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	21	APPC_ERR_INV_CONV_ID Invalid argument: CONVERSATION_ID
	27	APPC_ERR_INV_ERROR_ORIGIN Invalid argument: ERROR_ORIGIN
	80	APPC_ERR_UNEXPECTED_ERROR Unexpected error encountered.
	90	APPC_ERR_UNEXPECTED_DOS_ERROR Unexpected OS/2 error encountered.

```

Example:  # Subroutine error
          response to stimulus APPC_EVENT APPC_SystemError ConvID
          call APPCGetSystemError
            ( ConvID,           # ID from Start/AcceptStart
              ErrorIndicator,    # Work-request error
              WorkRequest,       # Work request ID
              APPCLastVERB,      # APPC Verb reporting error
              PrimaryReturnCode, # integer (6 digit number)
              SecondaryReturnCode ) # integer (8 digit number)
          if (errorlevel = ...) then ...

# Data received
response to stimulus APPC_EVENT APPC_Data ConvID
call APPCGetString
  ( ConvID,
    AnswerString,
    NoConvert )

:
if (Status = "error") then # If error in answer
  copy APPC_BadGet to BadDataRecv # Set argument value
  call APPCSendError
    ( ConvID,           # Send error notification
      BadDataRecv )    # APPC_BadGet
end if

response to stimulus APPC_EVENT APPC_ReceiveError ConvID
# Application-determined response

response to stimulus APPC_EVENT APPC_SendError ConvID
# Application-determined response

```

19.5.21 APPCSendString()

The **APPCSendString()** subroutine is called to send data to a partner. The sender can request that data be sent immediately, can request a confirmation, and can indicate that a reply from the partner is expected before more data can be sent.

Note: An **APPC_ConfirmRequest** event notification is signalled to your EASEL OS/2 EE program when the partner program requests confirmation. Your EASEL OS/2 EE program must respond by either calling **APPCSendConfirmed()** or **APPCSendError()**. If your program calls **APPCSendConfirmed()**, it generates a confirmed notification to the partner program. When the partner is an EASEL OS/2 EE program, this triggers its **response to stimulus ... APPC_Confirmed** response statement. If your program calls **APPCSendError()**, it generates an error notification to the partner program. When the partner is an EASEL OS/2 EE program, this triggers **response to stimulus ... APPC_ReceiveError** or **response to stimulus ... APPC_SendError**.

APPCSendString() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCSendString( integer:CONVERSATION_ID,  
                           string:STRING_VAR,  
                           integer:AUTO_CONVERT,  
                           integer:SEND_IND,  
                           integer:MORE_TO_SEND )  
  
    library "eslappc"
```

19.5.21.1 Arguments

CONVERSATION_ID is an integer variable containing the conversation identifier, or handle, for the APPC conversation returned by either **APPCStart()** or **APPCAcceptStart()**.

STRING_VAR is a string variable that specifies the data to send to the partner.

AUTO_CONVERT is an integer variable which indicates whether automatic conversion from ASCII to EBCDIC of the contents of STRING_VAR should be performed. This conversion uses the code page tables specified in the EASEL OS/2 EE APPC profile. Data from a host computer may be in EBCDIC, data on the workstation is in ASCII. If **APPC_Default** is specified no conversion will occur. Valid constants are:

APPC_NoConvert	(Default) Requests that conversion not be done.
APPC_Convert	Requests that the string be converted from ASCII to EBCDIC.

SEND_IND is an integer variable used to specify control over when data is actually sent to the partner. Valid constants are:

- APPC_None** (Default) Data is accumulated in a buffer to be sent when the buffer is full, or when another request causes the data to be sent.
- APPC_Flush** The data is sent immediately.
- APPC_Confirm** The data is sent immediately and a confirm notification is requested.

Note: A response to stimulus ...

APPC_ConfirmRequest response definition MUST be included in your EASEL OS/2 EE program if confirmation will be requested by its partner. *Otherwise the conversation will not proceed!* The response to stimulus ... **APPC_Confirmed** response definition is activated when the partner program sends your program a confirmation, in response to your confirmation request.

- APPC_SyncLevel** The data is sent based upon the synchronization level of the conversation, established when it was started. If the synchronization level was specified as **APPC_None**, the data is sent immediately. If specified as **APPC_Confirm**, the data is sent immediately and a confirm notification is requested.

MORE_TO_SEND is a integer constant that indicates whether your program has more data to send following the string being sent with this call. Valid constants are:

- APPC_MoreData** (Default) Indicates that this application has more data to send before it expects a response.
- APPC_ReadyToReceive** Indicates that the application expects to receive data from its partner before it has more data to send.

19.5.21.2 Call

A call to **APPCSendString()** is made according to the following syntax:

Model

```
call APPCSendString( CONVERSATION_ID,  
                     STRING_VAR,  
                     AUTO_CONVERT,  
                     SEND_IND,  
                     MORE_TO_SEND )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	18	APPC_ERR_CONVERSION_ERROR Error on character conversion.
	21	APPC_ERR_INV_CONV_ID Invalid argument: CONVERSATION_ID
	24	APPC_ERR_INV_AUTO_CONVERT Invalid argument: AUTO_CONVERT
	25	APPC_ERR_INV_SEND_IND Invalid argument: SEND_IND
	26	APPC_ERR_INV_MORE_TO_SEND Invalid argument: MORE_TO_SEND
	31	APPC_ERR_INV_STRING_VAR Invalid string variable.
	46	APPC_ERR_LGTH_EXCDS_PRF_BUFSIZE Invalid argument: BUFFER_LENGTH String length exceeds buffer size specified in profile.
	70	APPC_ERR_UNEXPECTED_EASEL_ERROR Unexpected error on call to EASEL OS/2 EE routine.
	80	APPC_ERR_UNEXPECTED_ERROR Unexpected error encountered.
	90	APPC_ERR_UNEXPECTED_DOS_ERROR Unexpected OS/2 error encountered.

```

Example:  # Response to data from partner
response to stimulus APPC_EVENT APPC_Data ConvID
  call APPCGetString
    ( ConvID,      # Handle from Start/AcceptStart
      GetMove      # String to hold partner's move
      NoConvert ) # Do not convert data
  if (...) then
    copy "YES" to OK

  :
else
  copy "NO" to OK
end if

# Respond to request for confirmation
response to stimulus APPC_EVENT APPC_ConfirmRequest ConvID
If (OK = "YES") then      # If understood move
  call APPCSendConfirmed
    ( ConvID )           # Say everything OK
  call APPCSendString
    ( ConvID,            # Handle from Start/AcceptStart
      ResponseMove,      # String containing next move
      NoConvert,         # Do not convert data
      Confirm,           # Confirm receipt
      RdytoRecv )       # This send expects reply
else
  # Otherwise,
  call APPCSendError      # indicate a problem
    ( ConvID,
      BadInput )         # APPC_BadGet
end if

# Resend last data
response to stimulus APPC_EVENT APPC_ReceiveError ConvID

:
# Reconstruct last response
call APPCSendString
  ( ConvID,              # Handle from Start/AcceptStart
    ResponseMove,        # String containing last move
    NoConvert,           # Do not perform conversion
    Confirm,             # Confirm receipt
    RdytoRecv )         # This send expects reply

```

19.5.22 APPCSetAcceptProfile()

The **APPCSetAcceptProfile()** subroutine is called to initialize or update profile information for a conversation that calls **APPCAcceptStart()**. This routine may be used with **APPCGetAcceptProfile()** to query the user for the profile values. The profile values are used by the transaction program which issues the APPC calls.

Note: A sample program to do profile maintenance, APPCPROF.EAL, has been provided.

APPCSetAcceptProfile() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCSetAcceptProfile( string:PROFILE_NAME,  
                                integer:ACTION,  
                                string:INVOCATION_CMD,  
                                integer:INACTIVITY_TIMEOUT,  
                                integer:BUFFER_SIZE,  
                                integer:ASCII_TABLE,  
                                integer:EBCDIC_TABLE )  
  
    library "eslappc"
```

19.5.22.1 Arguments

PROFILE_NAME is a string variable that specifies the name of the profile. This name *must* be the same name specified as the remotely attachable TP name in the OS/2 Extended Edition Communications Manager profiles. This argument is case sensitive.

ACTION is an integer variable that specifies the profile operation to be undertaken. Its values are:

APPC_CreateProfile	(Default) A new profile is being created. If a profile with the same name already exists, the existing profile will not be updated and errorlevel will be set to indicate that an error has occurred.
APPC_ReplaceProfile	An existing profile is being updated. If the profile does not exist, it will be created automatically.
APPC_DeleteProfile	An existing profile is being deleted. The profile entry represented by PROFILE_NAME will be deleted from the profile file.

INVOCATION_CMD is a string variable that specifies a command issued by the partner TP (as a result of a call to **APPCAcceptStart()**). This argument is specified when your EASEL OS/2 EE program is started by its partner dynamically. In this case the command name includes a complete file name and can include a fully-qualified path where appropriate, as shown in the following examples:

```
eslrun.exe myeslpgm
eslrun.exe d:\mypgms\myeslpgm
c:\ease1-ee\eslrun.exe chesspgm -d Ease1InvocationParm black
```

INACTIVITY_TIMEOUT is an integer variable that specifies a timer value, in seconds, allowing state switching to occur automatically when a conversation with authority to send is inactive for more than the specified number of seconds. This is a value that may require tuning; five seconds is a reasonable initial value. If set to zero, no check is made to see if the partner application wants to send data between sends. *This prevents automatic switching from being initiated by the partner!* If automatic switching is disabled, your program, when it has completed sending, must be sure to call the **APPCSendString()** subroutine with the MORE_TO_SEND argument set to **APPC_ReadyToReceive** or else call the **APPCReadyToReceive()** subroutine. Maximum value is 2,145,600.

BUFFER_SIZE is an integer between 0 and 65500 (default: 2048) that specifies the buffer size, in bytes, to be used to send and receive data. If specified as zero, a default buffer size of 2K bytes is used. The buffer size must be large enough to contain the largest number of characters or bytes of data to be sent in one call.

ASCII_TABLE is an integer value which represents the ASCII code page used for conversions. If specified as zero, the current system code page is used.

EBCDIC_TABLE is an integer value which represents the EBCDIC code page used for conversions. If specified as zero, the international code page, 500, is used.

19.5.22.2 Call

A call to **APPCSetAcceptProfile()** is made according to the following syntax:

Model

```
call APPCSetAcceptProfile( PROFILE_NAME,
                           ACTION,
                           INVOCATION_CMD,
                           INACTIVITY_TIMEOUT,
                           BUFFER_SIZE,
                           ASCII_TABLE,
                           EBCDIC_TABLE )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	8	APPC_ERR_ON_FILE_OPEN Error opening file.
	10	APPC_ERR_ON_FILE_WRITE Error writing file.
	12	APPC_ERR_OUT_OF_DISK_SPACE Out of disk space.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	32	APPC_ERR_INV_PROFILE_NAME Invalid argument: PROFILE_NAME
	38	APPC_ERR_INV_INACT_TIMEOUT Invalid argument: INACTIVITY_TIMEOUT
	39	APPC_ERR_INV_BUFFER_SIZE Invalid argument: BUFFER_SIZE
	40	APPC_ERR_INV_ASCII_TABLE Invalid argument: ASCII_TABLE
	41	APPC_ERR_INV_EBCDIC_TABLE Invalid argument: EBCDIC_TABLE
	42	APPC_ERR_INV_PROFILE_ACTION Invalid argument: ACTION
	43	APPC_ERR_INV_INVOC_COMMAND Invalid argument: INVOCATION_CMD

Example: response to start
 call APPCSetAcceptProfile
 (Profile, # "MYPROFILE" example name
 ProfileAction, # "APPC_ReplaceProfile"
 Invoke_CMD, # "ESLRUN.EXE MYES'.PGM"
 Timeout, # 5 five seconds
 Buffer_Size, # 0 use 2K byte buffer
 ASCII_Table, # 0 use current code page
 EBCDIC_Table) # 0 use code page 500

19.5.23 APPCSetInitProfile()

The **APPCSetInitProfile()** subroutine is called to initialize or update profile information for a conversation that calls **APPCStart()**. This routine can be used with **APPCGetInitProfile()** to query the user for the profile values. The profile values are used by the transaction program which issues the APPC calls.

Note: A sample program to do profile maintenance, APPCPROF.EAL, has been provided.

APPCSetInitProfile() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCSetInitProfile( string:PROFILE_NAME,  
                             integer:ACTION,  
                             string:LOCAL_LU_ALIAS_NAME,  
                             string:PARTNER_LU_ALIAS_NAME,  
                             string:PARTNER_TP_NAME,  
                             string:MODE_NAME,  
                             string:SECURITY_PARAMETERS,  
                             integer:INACTIVITY_TIMEOUT,  
                             integer:BUFFER_SIZE,  
                             integer:ASCII_TABLE,  
                             integer:EBCDIC_TABLE )  
  
    library "eslappc"
```

19.5.23.1 Arguments

PROFILE_NAME is a string variable that specifies the name of the profile. This name is the same name specified as a argument to the **APPCStart()** subroutine call. This argument is case sensitive.

ACTION is an integer variable that specifies the profile operation to be undertaken. Its values are:

- | | |
|----------------------------|--|
| APPC_CreateProfile | (Default) A new profile is being created. If a profile with the same name already exists, the existing profile will not be updated and errorlevel will be set to indicate that an error has occurred. |
| APPC_ReplaceProfile | An existing profile is being updated. If the profile does not exist, it will be created automatically. |
| APPC_DeleteProfile | An existing profile is being deleted. The profile entry represented by PROFILE_NAME will be deleted from the profile file. |

LOCAL_LU_ALIAS_NAME is a string variable that specifies the locally-known name for the logical unit (LU). This name was established during configuration of OS/2 Extended Edition Communications Manager on the

Create/Change Local APPC Logical Unit Profile panel. Its maximum length is 8 characters. It also must be identified in the Partner LU Profile.

PARTNER_LU_ALIAS_NAME is a string variable that specifies the name of a partner logical unit (LU) profile that was established during configuration. Its maximum length is 8 characters.

PARTNER_TP_NAME is a string variable that specifies the name of the partner transaction program (TP). This argument is case sensitive. If the partner is using OS/2 Extended Edition Communications Manager, this is the TP name specified in the partner's configuration using the Remotely Attachable Transaction Program Profile. Its maximum length is 64 characters. If the partner is invoking an EASEL OS/2 EE program, the partner TP name must be the same as the profile name specified in the **APPCAcceptStart()** subroutine call and the partner TP filename must be ESLAPPCA.EXE.

MODE_NAME is a string variable that specifies the mode name. The mode identifies the characteristics of the allocated session. It is identified in the Partner LU Profile during configuration. Its maximum length is 8 characters.

SECURITY_PARAMETERS is a string variable that specifies the parameters used to validate access to a partner transaction program and its resources. The following options may be used:

none	The conversation does not use security.
same	The user ID is verified locally.
pgm/USERID/PASSWORD	The partner logical unit (LU) validates access with the security information provided. Separate parameters with a forward slash (/). USERID is the ID of the local user; its maximum length is 10 characters. This argument is case sensitive. PASSWORD is the password of the local user; its maximum length is 10 characters. This argument is case sensitive.

INACTIVITY_TIMEOUT is an integer variable that specifies a timer value, in seconds, allowing state switching to occur automatically when a conversation with authority to send is inactive for more than the specified number of seconds. This is a value that may require tuning; five seconds is a reasonable initial value. If set to zero, no check is made to see if the partner application wants to send data between sends. *This prevents automatic switching from being initiated by the partner!* If automatic switching is disabled, your program, when it has completed sending, must be sure to call the **APPCSendString()** subroutine with the MORE_TO_SEND argument set to **APPC_ReadyToReceive** or else call the **APPCReadyToReceive()** subroutine. Maximum value is 2,145,600.

`BUFFER_SIZE` is an integer between 0 and 65500 (default: 2048) that specifies the buffer size, in bytes, to be used to send and receive data. If specified as zero, a default buffer size of 2K bytes is used. The buffer size must be large enough to contain the largest number of characters or bytes of data to be sent in one call.

`ASCII_TABLE` is an integer value which represents the ASCII code page used for conversions. If specified as zero, the current system code page is used.

`EBCDIC_TABLE` is an integer value which represents the EBCDIC code page used for conversions. If specified as zero, the international code page, 500, is used.

19.5.23.2 Call

A call to **APPCSetInitProfile()** is made according to the following syntax:

Model

```
call APPCSetInitProfile( PROFILE_NAME,  
                        ACTION,  
                        LOCAL_LU_ALIAS_NAME,  
                        PARTNER_LU_ALIAS_NAME,  
                        PARTNER_TP_NAME,  
                        MODE_NAME,  
                        SECURITY_PARAMETERS,  
                        INACTIVITY_TIMEOUT,  
                        BUFFER_SIZE,  
                        ASCII_TABLE,  
                        EBCDIC_TABLE )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	8	APPC_ERR_ON_FILE_OPEN Error opening file.
	10	APPC_ERR_ON_FILE_WRITE Error writing file.
	12	APPC_ERR_OUT_OF_DISK_SPACE Out of disk space.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.

32 **APPC_ERR_INV_PROFILE_NAME**
Invalid argument: PROFILE_NAME

33 **APPC_ERR_INV_LOC_LU_ALIAS_NAME**
Invalid argument: LOCAL_LU_ALIAS_NAME

34 **APPC_ERR_INV_PAR_LU_ALIAS_NAME**
Invalid argument:
PARTNER_LU_ALIAS_NAME

35 **APPC_ERR_INV_PAR_TP_NAME**
Invalid argument: PARTNER_TP_NAME

36 **APPC_ERR_INV_MODE_NAME**
Invalid argument: MODE_NAME

37 **APPC_ERR_INV_SEC_PARMS**
Invalid argument: SECURITY_PARAMETERS

38 **APPC_ERR_INV_INACT_TIMEOUT**
Invalid argument: INACTIVITY_TIMEOUT

39 **APPC_ERR_INV_BUFFER_SIZE**
Invalid argument: BUFFER_SIZE

40 **APPC_ERR_INV_ASCII_TABLE**
Invalid argument: ASCII_TABLE

41 **APPC_ERR_INV_EBCDIC_TABLE**
Invalid argument: EBCDIC_TABLE

42 **APPC_ERR_INV_PROFILE_ACTION**
Invalid argument: ACTION

50 **APPC_ERR_PROFILE_ALREADY_EXISTS**
Profile already exists when
APPC_CreateProfile specified.

Example: response to start
 call APPCSetInitProfile
 (Profile, # "MYPROF"
 ProfileAction, # "APPC_ReplaceProfile"
 MyLU, # "LU1" example name
 PartnerLU, # "LU4" example name
 ReceiveTPName, # "MYPROF"
 ModeName, # "TKNRING" example name
 NoSecurity, # "none"
 TimeOut, # 5 five seconds
 Buffer_Size, # 0 use 2K byte buffer
 ASCII_Table, # 0 use current code page
 EBCDIC_Table) # 0 use code page 500

19.5.24 APPCStart()

The **APPCStart()** subroutine is called to initiate a conversation with a program in another machine. An identifier (OS/2 handle) representing the conversation is returned. This identifier must be passed in the other EASEL OS/2 EE APPC subroutine calls.

APPCStart() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCStart( integer:CONVERSATION_ID,  
                    string:PROFILE_NAME,  
                    integer:SYNC_LEVEL,  
                    integer:STATE_SWITCH )  
  
    library "eslappc"
```

19.5.24.1 Arguments

CONVERSATION_ID is an integer variable that is set to an identifier for the conversation by this subroutine.

PROFILE_NAME is a string variable that specifies the name of the profile established with the **APPCSetInitProfile()** subroutine.

SYNC_LEVEL is an integer variable used to select the conversation synchronization level that your program and its partner use. See 19.3.4, "Confirmation Processing" on page 19-16. Its values are:

APPC_None	(Default) Specifies that the conversation cannot use APPC confirmation processing. It is an error to call APPCSendString() with the SEND_IND argument set to APPC_Confirm , or to call APPCRequestConfirm() or APPCSendConfirmed() .
------------------	--

APPC_Confirm	The conversation may use the APPC confirmation facility to synchronize processing and must respond to a request for confirmation from the partner.
---------------------	--

STATE_SWITCH is an integer variable used to indicate whether the automatic interface or the controlled interface is to be used. The automatic interface reduces the number of calls needed to send and receive data; state switching is automatic. The controlled interface provides full APPC flexibility to sophisticated applications. Valid constants are:

APPC_Automatic	(Default) The conversation will use the automatic interface.
-----------------------	--

APPC_Controlled	The conversation will use the controlled interface.
------------------------	---

19.5.24.2 Call

A call to **APPCStart()** is made according to the following syntax:

Model

```
call APPCStart( CONVERSATION_ID,  
                PROFILE_NAME,  
                SYNC_LEVEL,  
                STATE_SWITCH )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	22	APPC_ERR_INV_SYNC_LEVEL Invalid argument: SYNC_LEVEL
	23	APPC_ERR_INV_STATE_SWITCH Invalid argument: STATE_SWITCH
	32	APPC_ERR_INV_PROFILE_NAME Invalid argument: PROFILE_NAME
	70	APPC_ERR_UNEXPECTED_EASEL_ERROR Unexpected error on call to EASEL OS/2 EE routine.
	80	APPC_ERR_UNEXPECTED_ERROR Unexpected error encountered.
	90	APPC_ERR_UNEXPECTED_DOS_ERROR Unexpected OS/2 error encountered.
	100	APPC_ERROR Call APPCGetSystemError() to get error information returned in PRIMARY_ERROR_CODE and SECONDARY_ERROR_CODE.

Example: response to start
 call APPCStart
 (ConvID, # Conversation handle returned
 Profile, # "MYPROF"
 ConvSync, # APPC_Confirm
 AutoSwitch) # APPC_Automatic .

19.5.25 APPCTestReqToSend()

The **APPCTestReqToSend()** subroutine is called to query if the partner program wants to send data. If a request to send has been received from the partner, your program will then receive an **APPC_RequestToSend** event notification.

APPCTestReqToSend() is declared in the EASEL OS/2 EE APPC include file (**eslappc.inc**) according to the following syntax:

```
subroutine APPCTestReqToSend( integer:CONVERSATION_ID )  
    library "eslappc"
```

19.5.25.1 Arguments

CONVERSATION_ID is an integer variable containing the conversation identifier, or handle, for the APPC conversation returned by either **APPCStart()** or **APPCAcceptStart()**.

19.5.25.2 Call

A call to **APPCTestReqToSend()** is made according to the following syntax:

Model

```
call APPCTestReqToSend( CONVERSATION_ID )
```

<i>Returns:</i>	errorlevel	Error Integer Constant/Explanation
	0	APPC_NORMAL_COMPLETION Normal completion.
	16	APPC_ERR_OUT_OF_MEMORY Out of memory.
	21	APPC_ERR_INV_CONV_ID Invalid argument: CONVERSATION_ID
	80	APPC_ERR_UNEXPECTED_ERROR Unexpected error encountered.
	90	APPC_ERR_UNEXPECTED_DOS_ERROR Unexpected OS/2 error encountered.

Example: # A period of time has elapsed since last send
 call APPCTestReqToSend # Does partner want to send?
 (ConvID) # Handle from Start/AcceptStart

 # Received request
 response to stimulus APPC_EVENT APPC_RequestToSend ConvID
 call APPCReceiveNotify # Allow partner to send
 (ConvID) # Handle from Start/AcceptStart

19.6 Sample Code: Automatic Interface

A skeleton of a Chess program that cooperatively runs on two PCs.
The person who initiates the game is WHITE, the partner program
is passed "BLACK" as a parameter. The same program can initiate
the game or accept the initiation. When "BLACK" is passed as an
invocation parameter the program knows to do an APPCAcceptStart.

```
include "eslappc.inc"           # APPC include file
stimulus APPC_EVENT "eslappc"  # APPC_EVENT is a name for the
                                # set of messages/stimuli
                                # generated by the APPC DLL
```

```
# EASEL OS/2 EE program initialization
response to start
  if (InvocationParm = "BLACK") then
    call APPCAcceptStart
      ( ChessGame,           # Conversation handle returned
        AcceptProfileName,  # TP name = profile name
        SwitchStates )     # APPC_Automatic
  else
    # program will probably query the user for "what to start"
    call APPCStart
      ( ChessGame,           # Handle is returned
        ProfileName,        # "CHESS" example name
        SyncLevel,          # APPC_Default - no confirm
        SwitchStates )     # APPC_Automatic
    call APPCSendString
      ( ChessGame,           # Handle from Start/AcceptStart
        FirstMove,          # First move
        NoConvert,          # Do not convert string
        Force,              # Send data now
        PrepareToReceive )  # End of this send - expect reply
  end if
```

```
# Response to each partner move
response to stimulus APPC_EVENT APPC_Data ChessGame
  call APPCGetString
    ( ChessGame,           # Handle from Start/AcceptStart
      GetMove,             # String to receive Partner's move
      NoConvert )          # Do not convert string
```

```
:
```

```
call APPCSendString
  ( ChessGame,           # Handle from Start/AcceptStart
```

```

        ResponseMove,      # String containing next move
        NoConvert,         # Do not convert string
        Force,             # Send data now
        PrepareToReceive ) # This send expects reply

# Unrecoverable errors only
response to stimulus APPC_EVENT APPC_SystemError ChessGame
    call APPCGetSystemError
        ( ChessGame,      # Conversation id
          ErrorIndicator,  # Work request error
          WorkRequest,     # Last work request called
          APPCVerb,        # APPC verb number
          Status1,         # APPC error status1
          Status2 )        # APPC error status2
    call APPCEnd           # End this conversation
        ( ChessGame,      # Conversation id
          Abnormal )       # APPC_Abend
    copy 0 to ChessGame    # Indicate conversation ended
    # Display or record errors

# Partner ended conversation
response to stimulus APPC_EVENT APPC_Ended ChessGame
    call APPCEnd
        ( ChessGame,      # Free conversation resources
          Flush )         # Send end of conversation
    copy 0 to ChessGame    # Indicate conversation ended
    # Display or record errors

# EASEL OS/2 EE program is terminating
response to MainWindow on close
    if (ChessGame != 0) then # If conversation not ended,
        call APPCEnd        # end it now
        ( ChessGame,
          Flush )           # Send end of conversation
    end if

```

19.7 Sample Code: Controlled Interface

A skeleton of a Chess program that cooperatively runs on two PCs.
The person who initiates the game is WHITE, the partner program
is passed "BLACK" as a parameter. The same program can initiate
the game or accept the initiation. When "BLACK" is passed as an
invocation parameter the program knows to do an APPCAcceptStart.

```
include "eslappc.inc"           # APPC include file
stimulus APPC_EVENT "eslappc"  # APPC_EVENT is a name for the
                                # set of messages/stimuli
                                # generated by the APPC DLL

# EASEL OS/2 EE program initialization
response to start
    if (InvocationParm = "BLACK") then
        call APPCAcceptStart
            ( ChessGame,      # Handle is returned
              AcceptProfileName, # TP name = profile name
              SwitchStates ) # APPC_Controlled
        call APPCReceiveNotify
            ( ChessGame )    # Issue a receive
    else
        # program will probably query the user for "what to start"
        call APPCStart
            ( ChessGame,      # Handle is returned
              ProfileName,    # "CHESS"      example name
              SyncLevel,      # APPC_Default - no confirm
              SwitchStates ) # APPC_Controlled
        call APPCSendString
            ( ChessGame,      # Handle from Start/AcceptStart
              FirstMove,      # First move
              NoConvert,      # Do not convert string
              Force,          # Send data now
              RdyToReceive ) # End of this send - expect reply
        call APPCReceiveNotify
            ( ChessGame )    # Issue a receive
    end if
# EASEL OS/2 EE program is terminating
response to MainWindow on close
    if (ChessGame != 0) then # If conversation not ended
        call APPCEnd
            ( ChessGame,      # End it now
              Flush )        # Send end of conversation
    end if
```

```

# Start of Input
response to stimulus APPC_EVENT APPC_InputReceived ChessGame
begin ProcessInput

# Response to each move by partner
response to stimulus APPC_EVENT APPC_Data ChessGame
call APPCGetString
    ( ChessGame,      # Handle from Start/AcceptStart
      GetMove,        # String to hold partner's move
      NoConvert )     # Do not convert string

# Now in Send State
response to stimulus APPC_EVENT APPC_SendState ChessGame
call APPCSendString
    ( ChessGame,      # Handle from Start/AcceptStart
      ResponseMove,   # String containing next move
      NoConvert,      # Do not convert string
      Force,          # Send data now
      PrepareToReceive ) # This send expects reply

# Only unrecoverable errors
response to stimulus APPC_EVENT APPC_SystemError ChessGame
call APPCGetSystemError
    ( ChessGame,      # Conversation id
      ErrorCode,      # Work-request error code
      ID,             # ID of work request
      APPCVerb,       # Last APPC Verb called
      Status1,        # APPC error status1
      Status2 )       # APPC error status2
call APPCEnd
    ( ChessGame,      # End this conversation
      Abnormal )     # Abend conversation
copy 0 to ChessGame # Indicate conversation ended

:

# Display or record errors
# Partner ended conversation
response to stimulus APPC_EVENT APPC_Ended ChessGame
call APPCEnd
    ( ChessGame,      # Free conversation resources
      Flush)         # Send end of conversation
copy 0 to ChessGame # Indicate conversation ended

# End of input
response to stimulus APPC_EVENT APPC_EndReceive
if (ChessGame > 0) then
    # always want to receive response
    call APPCReceiveNotify
        ( ChessGame ) # Conversation id

```

```
    end if
  leave block
end ProcessInput
```

19.8 EASEL OS/2 EE APPC Reserved Constants

The following reserved constants are used in EASEL OS/2 EE APPC support. Be sure that your program does not contain any of these identifiers except when using them in the context of EASEL OS/2 EE APPC support.

Integer Constants Used to Set Argument Values for Subroutine Calls and as Stimulus Event Filters

APPC_Aband	APPC_Flush
APPC_Automatic	APPC_InputReceived
APPC_BadGet	APPC_MoreData
APPC_BadSend	APPC_NoConvert
APPC_Confirm	APPC_None
APPC_Confirmed	APPC_SendError
APPC_ConfirmRequest	APPC_ReadyToReceive
APPC_Controlled	APPC_ReceiveError
APPC_Convert	APPC_RequestToSend
APPC_CreateProfile	APPC_SendState
APPC_Data	APPC_ReplaceProfile
APPC_Default	APPC_Started
APPC_DeleteProfile	APPC_SyncLevel
APPC_Ended	APPC_SystemError
APPC_EndReceive	

EASEL OS/2 EE APPC Return Codes

APPC_ERROR	APPC_ERR_INV_PROFILE_NAME
APPC_ERR_BASIC_CONV_NOT_VALID	APPC_ERR_INV_SEC_PARMS
APPC_ERR_CONV_ENDING_REQ_PURGED	APPC_ERR_INV_SEND_IND
APPC_ERR_BUFFER_TOO_SMALL	APPC_ERR_INV_STATE_SWITCH
APPC_ERR_CONFIRM_REQ_IGNORED	APPC_ERR_INV_STRING_VAR
APPC_ERR_INFO_NOT_AVAILABLE	APPC_ERR_INV_SYNC_LEVEL
APPC_ERR_INVALID_PARMS	APPC_ERR_INV_TO_ISSUE_CONFIRMED
APPC_ERR_INV_ASCII_TABLE	APPC_ERR_INV_TO_ISSUE_CNFRM_REQ
APPC_ERR_INV_AUTO_CONVERT	APPC_ERR_INV_TO_ISSUE_RECV_NTFY
APPC_ERR_INV_BUFFER_LENGTH	APPC_ERR_INV_WINDOW_HDL
APPC_ERR_INV_BUFFER_POINTER	APPC_ERR_LGTH_EXCDS_PRF_BUFSIZE
APPC_ERR_INV_BUFFER_SIZE	APPC_ERR_NO_DATA_AVAILABLE
APPC_ERR_INV_CONV_ID	APPC_ERR_NO_REMOTE_TP_PARAM
APPC_ERR_INV_EBCDIC_TABLE	APPC_ERR_NO_SYSTEM_ERROR_DATA
APPC_ERR_INV_ERROR_ORIGIN	APPC_ERR_ON_FILE_OPEN
APPC_ERR_INV_INACT_TIMEOUT	APPC_ERR_ON_FILE_READ
APPC_ERR_INV_INDEX_END	APPC_ERR_ON_FILE_WRITE
APPC_ERR_INV_INDEX_START	APPC_ERR_OUT_OF_DISK_SPACE
APPC_ERR_INV_INTEGER_VAR	APPC_ERR_OUT_OF_MEMORY
APPC_ERR_INV_INVOC_COMMAND	APPC_ERR_PROFILE_ALREADY_EXISTS
APPC_ERR_INV_LOC_LU_ALIAS_NAME	APPC_ERR_STRING_TRUNCATED

APPC_ERR_INV_MODE_NAME
APPC_ERR_INV_MORE_TO_SEND
APPC_ERR_INV_PAR_LU_ALIAS_NAME
APPC_ERR_INV_PAR_TP_NAME
APPC_ERR_INV_PROFILE_ACTION

APPC_ERR_SYSTEM_ERROR_PENDING
APPC_ERR_UNEXPECTED_DOS_ERROR
APPC_ERR_UNEXPECTED_EASEL_ERROR
APPC_ERR_UNEXPECTED_ERROR

Values for Argument **Work_Request_Identifier()** in Subroutine **APPCGetSystemError()**

APPC_AcceptStart
APPC_ConvertString
APPC_GetInfo
APPC_ReceiveBuffer
APPC_ReceiveNotify
APPC_ReceiveString
APPC_RequestConfirm
APPC_RequestRdyToReceive
APPC_RequestSend

APPC_SendAbend
APPC_SendConfirmed
APPC_SendData
APPC_SendErrorItem
APPC_SendFlush
APPC_SendTerminate
APPC_Start
APPC_TestReqSend

Values for Argument **APPC_Verb** in Subroutine **APPCGetSystemError()**

AP_M_ALLOCATE
AP_M_CONFIRM
AP_M_CONFIRMED
AP_M_DEALLOCATE
AP_M_FLUSH
AP_M_GET_ATTRIBUTES
AP_M_PREPARE_TO_RECEIVE
AP_M_RECEIVE_AND_POST

AP_M_RECEIVE_AND_WAIT
AP_M_RECEIVE_IMMEDIATE
AP_M_REQUEST_TO_SEND
AP_M_SEND_DATA
AP_M_SEND_ERROR
AP_M_TEST_RTS
AP_RECEIVE_ALLOCATE
AP_TP_ENDED
AP_TP_STARTED
SV_GET_CP_CONVERT_TABLE

19.9 Communications Manager Considerations

The following information is provided to assist you in configuring certain areas of OS/2 Extended Edition Communications Manager so that EASEL OS/2 EE APPC will operate correctly with CM on your machine. The information provided below is intended only as a supplement to, not as a replacement for, the more complete and detailed CM profile configuration information found in the OS/2 Extended Edition documentation. Please refer to the *IBM Operating System/2 Extended Edition User's Guide* for detailed CM configuration information.

19.9.1 Configuring CM for an EASEL OS/2 EE Acceptor

If your EASEL OS/2 EE APPC program calls the **APPCAcceptStart()** sub-routine, then it is considered an acceptor program and you must configure a CM profile for the EASEL OS/2 EE APPC acceptor TP (ESLAPPCA.EXE). Here's how to do it:

1. On the "Communications Manager Main Menu", select the Advanced action bar choice, then select Configuration... from the pulldown menu.
2. Specify the configuration file name in the pulldown.
3. On the "Communication Configuration Menu", select SNA feature profiles
4. On the "SNA Feature Configuration" menu, select APPC remotely attachable transaction program (TP) profiles...
5. On the Profile Operations pulldown, select Create...
6. Specify the name of this CM profile you are creating in the pulldown.
7. On the "Create/Change Remotely Attachable Transaction Program Profile" panel, supply the following parameter values:

TP filespec	Specify the full path and filename with extension for the EASEL OS/2 EE APPC acceptor TP (ESLAPPCA.EXE). For example, specify:
-------------	--

c:\easel-ee\eslapcca.exe

Service TP	Select No.
------------	------------

Sync level	All options are valid; select whichever one is appropriate for your conversation. If your program does not support confirmation processing, select None.
------------	--

Conversation type	Select Mapped.
-------------------	----------------

Conversation security	All options are valid; select whichever one is appropriate for your conversation. Your answer here should
-----------------------	---

be consistent with the `SECURITY_PARAMETERS` value you set using **`APPCSetInitProfile( )`**.

TP operation All options are valid.

If you select

Queued - operator started...,

then the **`APPCAcceptStart( )`** issued in your EASEL OS/2 EE APPC program will correctly start the TP. Your program should be started before an allocate request arrives from its partner; your program will wait for the **`APPC_Started`** event notification generated by the allocate request.

If you select

Non-queued - attach manager started..., then you must also specify the EASEL OS/2 EE APPC acceptor profile name in the TP start-up parameters input field on the Queued - Attach Manager Started pulldown that comes later in this profile configuration. That name is case sensitive. The EASEL OS/2 EE acceptor profile, which is built through **`APPCSetAcceptProfile( )`**, must specify in the `INVOCATION_CMD` argument how to start your EASEL OS/2 EE APPC program.

8. On the Specify TP Name pulldown, specify a TP name that is the same name as the EASEL OS/2 EE APPC acceptor profile name. This name is case sensitive. Maximum length is 32 characters.
9. On the Queued - Attach Manager Started or Non-Queued - Attach Manager Started dialog box, supply the following parameter values:

TP start-up parameters

Specify the EASEL OS/2 EE APPC acceptor profile name. This name is case sensitive.

Program type Specify Presentation Manager (windowed)

19.9.2 Enabling APPC Tracing

Tracing may prove helpful in debugging your EASEL OS/2 EE APPC applications. The trace shows the APPC verbs issued, the parameters passed and returned, and the return codes.

Here's how to turn tracing on:

1. On the *Communications ManagerMain Menu*, select the *Advanced* action bar choice, then select *Problem determination aids* from the pulldown menu.
2. On the *Problem Determination Aids* menu, select *Trace services*.
3. On the *Trace Services* menu, select *Select traces*.
4. On the *Trace Type Select* panel, select *Trace Selection*.

5. On the *Trace Selection* panel, under the column *APIs*, select *APPC*.
6. When the *Trace Services* menu returns, select *Start selected traces*.

Note: You must stop tracing in order to enable the option to copy the trace output to a file.

Chapter 20. 3270 Support

20.1 Introduction

20.1.1 EASEL OS/2 EE in the 3270 Environment

3270 is the generic name of a family of IBM terminals, controllers, and associated software. 3270 applications transmit information on a screen-by-screen basis. With 3270 Support for EASEL OS/2 EE, a full screen of information is held in a screen buffer, where its data can be accessed. 3270 Support for EASEL OS/2 EE provides facilities that allow you to examine data in the buffer, place data into specified fields, transfer data to the host, and execute other functions that can be performed by a 3270 terminal. These functions are accomplished by calling **M3270** module action routines described in 20.4, "M3270 Module Action Routines" on page 20-13.

20.1.2 Components of 3270 Support for EASEL OS/2 EE

The following 3270 Support for EASEL OS/2 EE components allow you to create an EASEL OS/2 EE interface to a 3270 application:

1. An EASEL OS/2 EE module named **M3270**.

This module defines EASEL OS/2 EE action routines specific to the 3270 environment and EASEL OS/2 EE variables used by the 3270 action routines. These action routines:

- Read the contents of the screen buffer into an EASEL OS/2 EE textual region.
- Read a line into a string variable.
- Read a field's contents into a string variable.
- Enter characters into a screen buffer field.
- Send data to the controller.
- Simulate the pressing of a 3270 terminal's special function keys.

2. A local application named **A3270.EXE**.

EASEL OS/2 EE 3270 Support uses this application internally, to communicate with 3270 host applications via Communications Manager (CM) and EHLLAPI.

3. An EASEL OS/2 EE development tool named **FIELDS.EXE**.

This tool provides you with essential information about each field on the current screen in the 3270 screen buffer.

20.1.3 Using the M3270 Module: Required Procedures

The following steps are required in order to successfully implement an EASEL OS/2 EE program which uses the **M3270** module.

1. Install the IBM OS/2 Extended Edition CM, if it is not already installed.
2. Make sure 3270 terminal emulation is started before running an EASEL OS/2 EE program that uses the **M3270** module or before running the **FIELDS** utility. 3270 terminal emulation can be started from the CM "Start Communications" menu (or, CM can be configured to start 3270 communications automatically when it is loaded). Refer to the IBM Operating System/2 Extended Edition User's Guide, "Using Communications Manager" for more information.
3. Where necessary, examine the output generated by the **FIELDS.EXE** tool, to identify the characteristics that tell when a screen is in the buffer and the precise contents of the buffer.
4. Write a test program that:
 - a. References the **M3270** module (see below).
 - b. Initializes 3270 communications and connects to one or more sessions.
 - c. Synchronizes the timing of 3270 and EASEL OS/2 EE via **M3270** action routines, so that the screen buffer is filled before actions on it are executed on it.
 - d. Reads specific buffer contents into EASEL OS/2 EE string variables and textual regions, and communicates with the host via **M3270** action routines.
5. Write the final program, incorporating step 4 above with other EASEL OS/2 EE action and response statements.

20.1.4 Referencing the M3270 Module

To reference the **M3270** module from an EASEL OS/2 EE program, include the following statement at the very beginning of the program:

```
module M3270
```

20.1.5 Sample Program

The following example shows a complete EASEL OS/2 EE program for a 3270 application that contains one full screen buffer.

```
module M3270                                # References the M3270 module.

primary textual region Report               # Defines a textual region in which to
  window size 79 columns 24 lines          # display the data.
  at position 0 100
  title bar "Report"
  system menu
  white border
  font "asc814"

response to start
  copy "A" to SessionName                  # Initial session to connect.
  action Init3270                          # Initializes 3270 communications.
  copy "Report" to TextRegionName          # Copies the name of the textual regi
                                          # that will store the screen data to
                                          # TextRegionName (a predefined
                                          # variable).
  action ReadScreen                        # Copies contents of the 3270 screen
                                          # buffer into the textual region.

response to Report on close
  action Disconnect                        # Disconnects from session A.
  action Stop3270                          # Stops 3270 communications.
  exit                                    # Exits EASEL OS/2 EE.
```

20.2 3270 Session and Screen Concepts

20.2.1 Logical Terminal Sessions

CM allows you to create up to five logical 3270 terminal sessions. A work station can emulate up to five 3270 display terminals at one time. **M3270** action routines allow EASEL OS/2 EE to be in control of one or more of these sessions.

Each session configured by CM has two names, a long session name and a short session name. The long session name is up to eight characters long. The short session name is a single letter, such as "A" or "B." The **M3270** action routines use only short session names to identify sessions. 5250 host sessions cannot be accessed by **M3270** action routines, but both 5250 and 3270 host sessions can be accessed in the same program by using the **M5250** and **M3270** action routines, respectively.

From the point of view of EASEL OS/2 EE, each configured session, once it has been started by CM, can be in one of three states:

1. Disconnected

The session is not controlled by EASEL OS/2 EE. It may, however, be currently controlled by another process.

2. Connected, not active

The session is controlled by EASEL OS/2 EE. However, it is not the session acted upon by the set of **M3270** action routines which read, write, or send data. Other running processes are prevented from connecting to this session.

3. Connected and active

The session is controlled by EASEL OS/2 EE and is acted upon by the set of **M3270** action routines which read, write or send data. Other running processes are prevented from connecting to this session.

At any given time, EASEL OS/2 EE may be connected to none, some, or all of the started sessions. If it is connected to one or more sessions, only one of these may be the active connected session; the rest, if any, are not active. If there is no active connected session, an error condition will result from calling any **M3270** action routine which reads, writes, or sends data, returns status information, etc.

The **M3270** action routines which connect to or disconnect from sessions are **Init3270**, **Stop3270**, **Connect**, and **Disconnect**.

20.2.2 Screen Sizes

Each session is configured with one of the following screen sizes:

Model Number	Number of Lines	Number of Columns
Mod 2	24	80
Mod 3	32	80
Mod 4	43	80
Mod 5	27	132

The Model number refers to the emulated 3270 terminal model whose display is of the given dimensions. Note that the numbers of lines shown in this table do not include the Operator Information Area displayed on the bottom line of the terminal.

Whenever EASEL OS/2 EE connects to a session, it automatically determines the screen size with which the session was configured.

20.2.3 Field-oriented and Line-oriented Screens

The 3270 *screen buffer* is generally *field-oriented* (also called *formatted*), or organized by fields.

Most **M3270** action routines are designed for accessing field-oriented screens. 3270 Support for EASEL OS/2 EE accesses the screen buffer by first assigning sequential numbers to each field in the buffer. By specifying a field number, you can retrieve that field's contents and associated attributes, or write text into the specified field.

A 3270 application sometimes contains screens that have no fields at all. These are called *line-oriented* (or *unformatted*) screens. There are two **M3270** action routines used to access line-oriented screens: **ReadLine**, and **FindLastNonBlankLine**. You can use these action routines to access line-oriented or field-oriented screens, but you cannot use the field-oriented action routines to access line-oriented screens.

20.2.3.1 Field-oriented (Formatted) Screens

By definition, a field is at least one character in length. A field can wrap around to the next line, or even to the beginning of the screen buffer. Each field is preceded by a *field attribute character*, which defines the location and attributes of the field, but is not part of the field itself. The following attributes are defined by the field attribute character:

Attribute	Description
protected/unprotected	A protected field cannot be modified (written) by the user.

modified/unmodified	Indicates whether the field was modified (written).
alphanumeric/numeric	Indicates whether an unprotected field is intended for alphanumeric or numeric data.
display/non-display	Indicates whether the field is displayed on the 3270 terminal.
intensified/normal	Indicates brightness of the characters in the field.

Each field extends up to the next field attribute character, which defines the next field in the screen.

The 3270 application may also assign *extended attributes* to individual characters in the screen buffer. Of these, EASEL OS/2 EE supports the extended attributes for **color**, so that you can examine individual characters for color, once they have been read into a colored textual region.

Character positions within the 3270 buffer are addressed by row and column. Row numbers go from 1 to the number of lines in the screen, from top to bottom; column numbers go from 1 to the number of columns in the screen, from left to right. The following examples illustrate how a character position is expressed as an ordered pair of numbers:

(1,1) = row 1, column 1
 (24,80) = row 24, column 80
 (43,80) = row 43, column 80 (mod 4 screen only)

Characters within the 3270 Operator Information Area (the bottom line of the host system) cannot be directly accessed via 3270 support for EASEL OS/2 EE (see **GetXstatus** and **GetSessionStatus**).

20.3 Retrieving Field Information with **FIELDS.EXE**

FIELDS.EXE is a development tool which generates a report that contains up to three sections:

1. Information about the current screen:
 - Index by column number
 - Entire screen contents
 - Number of fields in the screen
 - Current cursor position, by row and column.
2. A field summary, including the following information for each field in the screen:
 - Field number
 - Starting and ending field positions, by row and column
 - Field length
 - Field attributes
 - First 26 characters in the field.
3. Full field contents, listing all characters in each field.

By default, only the first two sections are reported. In general, the field summary section provides all of the field information you need to develop an EASEL OS/2 EE program. (See 20.3.1, "FIELDS.EXE Sample Output" on page 20-10.)

To use **FIELDS.EXE**:

1. Access the 3270 host application (using the CM windowed 3270 terminal emulator) and go to the screen for which you need the field information.
2. Activate an OS/2 fullscreen or windowed command prompt session.
3. At the OS/2 command prompt, invoke **FIELDS.EXE** as follows:

Model

fields [OPTIONS] [OUTPUT_FILENAME]

Where: OPTIONS is a string containing any of the following parameters:

Parameter	Meaning
-----------	---------

-sNAME	Report is for the terminal session represented by the specified single-letter short session name. This parameter defaults to the first session listed by FIELDS.EXE when "-L" is specified. It is recommended that you always specify this parameter when CM is configured for more than one session.
-b	Report contains section on screen information only.
-f	Report contains all three sections.
-L	Report lists all 3270 and 5250 host sessions started by CM. No field information is given. For each session, its long name, short name and screen dimensions are shown. Sessions that have not been started are not listed.

OUTPUT_FILENAME

The name and extension of the ASCII text file in which the FIELDS report is placed. If OUTPUT_FILENAME is omitted, the report is listed on your workstation's display, and the screen information section does not show the screen contents.

Note: If "-L" is specified, this parameter is ignored; the sessions are always listed on your workstation's display.

Examples:

```
fields -sA
fields login.fld
fields -b news4.fld
fields -f -sB product.fld
```

20.3.1 FIELDS.EXE Sample Output

The following is the sample output for the **-L** parameter:

FIELDS utility - Version E1.1 (EHLLAPI)
International Business Machines Corporation
(C) Copyright Interactive Images, Inc. 1987, 1990
All Rights Reserved.

Number of started host sessions = 4

Short	Long			
<u>Name</u>	<u>Name</u>	<u>Rows</u>	<u>Columns</u>	<u>Type</u>
A	HOST1	24	80	3270 Host
B	HOST2	32	80	3270 Host
C	HOST3	43	80	3270 Host
D	HOST4	24	80	5250 Host

The following two pages show a sample output from **FIELDS.EXE**, when no listing options are specified. The area shown within the double dashed lines is how the entire screen would appear on a terminal. The first double dashed line is preceded by a column number index.

FIELDS utility - Version E1.1 (EHLAPI)
International Business Machines Corporation
(C) Copyright Interactive Images, Inc. 1987, 1990
All Rights Reserved.

Session short name = A

00000000011111111222222222233333333334444444445555555556666666667777777778
1234567890123456789012345678901234567890123456789012345678901234567890
=====

SVM1234P PRIMARY SERVICE SELECTION MENU
SYSTEM: IIIIOAQW2
TERMID: IIIIZXCV3
CUSTOMER ASSISTANCE: ENTER "NOTIFY" OR CALL 617-938-8440

TIME: 17:46:59

- NAME OF SERVICE
- 1 MESsage
 - 2 NEWs
 - 3 NOTIfy
 - 4 PROfile

Enter selection or press the HELP key (PF1) for assistance

PF1=HELP PF3=END

==>

=====

Screen contains 29 fields.
Cursor is at [24,7].

Field summary:

Unp = Unprotected
Num = Numeric

Int = Intensified
Nds = Nondisplay

Mod = Modified

Fld#	Start	End	Length	Unp	Num	Int	Nds	Mod	First 26 Characters
1	[1,2]	[1,21]	20		X				[SVM1234P]
2	[1,23]	[1,80]	58		X	X			[PRIMARY SERVICE SELECTION
3	[2,2]	[2,8]	7		X				[SYSTEM:]
4	[2,10]	[2,63]	54		X				[IIIOAQW2
5	[2,65]	[2,69]	5		X				[TIME:]
6	[2,71]	[2,80]	10		X				[17:46:59]
7	[3,2]	[3,8]	7		X				[TERMID:]
8	[3,10]	[3,80]	71		X				[IIIZXCV3
9	[4,2]	[4,44]	43		X				[CUSTOMER ASSISTANCE: ENTER
10	[4,46]	[7,5]	200		X				[617-938-8440
11	[7,7]	[8,2]	76		X	X			[NAME OF SERVICE
12	[8,4]	[8,9]	6		X				[1]
13	[8,11]	[9,2]	72		X				[MESSAGE
14	[9,4]	[9,9]	6		X				[2]
15	[9,11]	[10,2]	72		X				[NEWS
16	[10,4]	[10,9]	6		X				[3]
17	[10,11]	[11,2]	72		X				[NOTify
18	[11,4]	[11,9]	6		X				[4]
19	[11,11]	[12,2]	72		X				[PASSword
20	[12,4]	[12,9]	6		X				[5]
21	[12,11]	[18,80]	550		X				[PROFILE
22	[19,2]	[20,80]	159		X	X			[Enter selection or press t
23	[21,2]	[21,80]	79		X				[PF1=HELP PF3=END
24	[22,2]	[23,80]	159		X				[
25	[24,2]	[24,5]	4		X				[====]
26	[24,7]	[24,14]	8	X				X	[]
27	[24,16]	[24,69]	54		X				[
28	[24,71]	[24,78]	8	X				X	[]
29	[24,80]	[24,80]	1		X				[]

20.4 M3270 Module Action Routines

The **M3270** module defines a set of variables and action routines. Many action routines are associated with two sets of variables: *input* variables and *output* variables. By copying values to the input variables of a specified action routine and then calling the action routine, data is returned in the output variables of the action routine.

Other action routines have only input variables or only output variables. Some action routines have neither.

In the example below, the program reads a street address contained in the third field of the 3270 screen buffer and displays it as part of a graphical object on the EASEL OS/2 EE screen.

```
copy 3 to FieldNumber    # input variable for ReadField
action ReadField          # call action routine
change AddressKey to     # AddressKey is a graphical
    text FieldText       # object output variable
                        # from ReadField
```

20.4.1 M3270 Action Routines and EASEL OS/2 EE Built-in Functions

All **M3270** action routines except the following reset all EASEL OS/2 EE response inquiry built-in functions:

DefineWatch	WatchForCursorAt
WatchForCursorNotAt	WatchForNoX
WatchForChar	WatchForNotChar

For example:

```
response to Key_X        # This will not work,
    action PressENTER    # because "object" was
    make object green    # reset by "PressENTER."
```

If you need to preserve the value of an EASEL OS/2 EE response inquiry built-in function, copy the value to a temporary variable before calling any **M3270** action routines. Then the temporary variable name can be used whenever the built-in function value is required. The following example preserves the value of the **object** built-in function in the string variable Temp.

```
copy object to Temp
:
action ...
:
make Temp invisible
```

20.4.2 Summary of M3270 Action Routines

The following table summarizes the action routines provided in the **M3270** module. Following the summary, each action is described in detail.

Action Routine	Usage
<i>Starting/Stopping:</i>	
Init3270	Initializes M3270 variables, starts A3270 local application, and connects to a session.
Stop3270	Disconnects all connected sessions and stops A3270 local application.
<i>Connecting and Disconnecting Sessions:</i>	
Connect	Connects to a specified session.
Disconnect	Disconnects from the active connected session.
<i>Examining the Buffer:</i>	
ScanScreen (Field-oriented)	Records locations and sizes of all fields in the 3270 screen buffer; returns a variable containing the number of fields.
ReadField (Field-oriented)	Copies contents of specified field into a string variable.
GetFieldLength (Field-oriented)	Returns the length of a specified field.
GetFieldAttributes (Field-oriented)	Returns booleans based on field attributes (Protected , Alphanumeric , Modified , NonDisplay , Intensified) of a specified field.
GetFieldNumber (Field-oriented)	Locates the 3270 field containing the specified screen position.
GetCursorPosition	Returns row and column of current cursor position.
GetFieldPosition (Field-oriented)	Returns the starting row and column position of a specified field.
ReadLine (Line-oriented)	Copies contents of a line into a string variable.
FindLastNonBlankLine (Line-oriented)	Returns the last nonblank line at or above a specified starting line.
ReadScreen	Inserts the contents of the 3270 screen buffer into a textual region.
ReadColoredScreen	Inserts the contents of the 3270 screen buffer, with colors, into a colored textual region.

Synchronization:

Action Routine	Usage
DefineWatch	Is called in preparation for one or more of the following “watch for” action routines to be called.
WatchForCursorAt	Queues a command for watching the 3270 screen until the cursor is located at a specified row and column position.
WatchForCursorNotAt	Queues a command for watching the 3270 screen until the cursor is located anywhere except a specified row and column position.
WatchForNoX	Queues a command for watching the Operator Information Area until the “X” is gone and remains gone for a specified period of time.
WatchForChar	Queues a command for watching the 3270 screen until a specified character appears at a specified row and column position.
WatchForNotChar	Queues a command for watching the 3270 screen until any character other than a specified character appears at a specified row and column position.
Watch	Executes previously queued “watch” command(s) and activates a response you have defined to signal when a new screen is ready. Allows EASEL OS/2 EE to take other responses while “watch” is in effect.
WatchAndWait	Executes previously queued “watch” command(s) and waits until a new screen is ready before allowing EASEL OS/2 EE to take other responses.

Communicating with the Host:

PressKey	Emulates pressing a specified “special” key on a 3270 terminal.
PressENTER	Emulates pressing the ENTER key on a 3270 terminal.
PressPFkey	Emulates pressing a PF key on a 3270 terminal.
TypeString	Emulates pressing keys corresponding to any of the printing ASCII characters.
EnterString	Same as TypeString followed by PressENTER.

Controlling the Emulator Window:

SetEmulatorWindow	Changes the state of the emulator window.
--------------------------	---

Action Routine	Usage
Emulate3270	Activates emulator window. Returns when user minimizes emulator window.
EmulateAndWatch	Similar to Emulate3270 . Returns when emulator displays a screen defined in a screen-ID file.
<i>Miscellaneous Action Routines:</i>	
WriteField (Field-oriented)	Copies a string into a specified field.
GetXstatus	Returns the condition of "X" in the Operator Information Area.
Print3270Screen	Prints the current 3270 screen buffer contents. Used for debugging.
GetSessionStatus	Returns readiness status and system connection information of the current session.
GetSessionInfo	Returns session type and screen dimensions of a specified session.

Each action routine is discussed below in the same order as they are introduced above. The description lists the action routine's associated input variables and output variables, if any, notes error conditions, gives sample EASEL OS/2 EE code where applicable, and refers you to other associated action routines.

20.4.3 Starting and Stopping 3270 Support for EASEL OS/2 EE

The following two action routines start and stop A3270, the local application that allows EASEL OS/2 EE to communicate with 3270 sessions.

20.4.3.1 Init3270

Syntax: **action Init3270**

Input **SessionName**

Variables:

Output None

Variables:

Remarks: **Init3270** copies initial values to global variables used by the other **M3270** action routines, starts the local application A3270, and connects to the session specified in **SessionName**.

 Use action **Connect** if you need to connect to additional sessions.

Example: response to start
 # Initialize 3270 communications and connect to
 # Session B.
 copy "B" to SessionName
 action Init3270

Errors: E_CANTINIT Cannot initialize
 E_SESSNAME Invalid session name
 E_INUSE Session is already in use

See Also: **Stop3270 Connect**

20.4.3.2 Stop3270

Syntax: **action Stop3270**

Input None

Variables:

Output None

Variables:

Remarks: **Stop3270** should be called when your program has completed 3270 communications. You should ensure that all connected sessions get logged off. Although **Stop3270** will disconnect all connected sessions, it is good programming practice to call action **Disconnect** for each connected session first.

Example: response to StopSign
 copy "LOGOUT" to Keystrokes
 action EnterString
 action Disconnect
 action Stop3270
 exit

See Also: **Disconnect**

20.4.4 Connecting and Disconnecting Sessions

20.4.4.1 Connect

Syntax: **action Connect**

Input **SessionName**

Variables:

Output None

Variables:

Remarks: EASEL OS/2 EE connects to the session specified by **SessionName**. This session becomes the active connected session. The previous active connected session, if any, remains connected, but it is no longer active. It is not an error to specify a session which is already connected; in fact, this is how you reactivate it.

Example: copy "A" to SessionName
 action Connect
 # Session A is now the active connected session.

 copy "B" to SessionName
 action Connect
 # Session A is connected; session B is the active
 # connected session.

 copy "A" to SessionName
 action Connect
 # Session B is connected; session A is the active
 # connected session.

Errors: E_SESSNAME Invalid session name
 E_INUSE Session is already in use

20.4.4.2 Disconnect

Syntax: **action Disconnect**

Input None

Variables:

Output None

Variables:

Remarks: **Disconnect** disconnects from the active connected session. To disconnect from a session which is connected but not active, you must first make it active by calling action **Connect**. When action **Disconnect** returns, there is no active connected session (although there may still be connected sessions).

Example: # Disconnect from sessions A and B. The current
 # active connected session is A.
 action Disconnect # Disconnect from A.
 copy "B" to SessionName
 action Connect # Make B active.
 action Disconnect # Disconnect from B.

Errors: E_NOTCONN Not connected to an active session

20.4.5 Examining the Buffer

The following action routines allow you to determine the contents of the screen buffer for the active connected session.

20.4.5.1 ScanScreen

Syntax: **action ScanScreen**

Input None

Variables:

Output **FieldCount** (integer)
Variables:

Remarks: **ScanScreen** scans the screen buffer and records the locations and sizes of all the fields. Upon completion of this routine, the output variable **FieldCount** contains the total number of fields in the screen buffer. You cannot directly access the field information that is recorded, but the information is available through and is used by many of the action routines described in this section.

You must call **ScanScreen** after the contents of the screen have changed, if the new screen's format is different and if you are referencing this screen by fields.

Once **ScanScreen** is called, any field can be specified by setting the variable **FieldNumber** to a value between **1** and **FieldCount**, inclusive.

Example: response to start
 copy "C" to SessionName
 action Init3270
 action ScanScreen
 :

20.4.5.2 ReadField

<i>Syntax:</i>	action ReadField	
<i>Input Variables:</i>	FieldNumber	(integer)
<i>Output Variables:</i>	FieldText	(string)
<i>Remarks:</i>	ReadField copies the contents of the field specified by FieldNumber into the string variable FieldText . You can use ReadField to examine the contents of non-display fields.	
<i>Errors:</i>	E_NOFLDS	Screen contains no fields or has not been scanned
	E_FLDNUM	Invalid field number
<i>Example:</i>	<pre># In this example, Date_Window is a textual region # defined in the program. : copy 4 to FieldNumber action ReadField add to Date_Window insert FieldText</pre>	

20.4.5.3 GetFieldLength

<i>Syntax:</i>	action GetFieldLength	
<i>Input</i>	FieldNumber	(integer)
<i>Variables:</i>		
<i>Output</i>	FieldLength	(integer)
<i>Variables:</i>		
<i>Remarks:</i>	GetFieldLength returns the length of a specified field. It will return zero if there is an error.	
<i>Example:</i>	copy 9 to FieldNumber action GetFieldLength if (FieldLength = 8) then : end if	
<i>Errors:</i>	E_NOFLDS	Screen contains no fields or has not been scanned
	E_FLDNUM	Invalid field number

20.4.5.4 GetFieldAttributes

Syntax: **action GetFieldAttributes**

Input **FieldNumber** (integer)
Variables:

Output **Protected** (boolean)
Variables: **Alphanumeric** (boolean)
 Modified (boolean)
 NonDisplay (boolean)
 Intensified (boolean)

Remarks: **GetFieldAttributes** sets the above output boolean variables to “true” or “false,” based upon the attribute character associated with the given field. All the booleans return “false” if there is an error.

Example: action ScanScreen
 copy 6 to FieldNumber
 action GetFieldAttributes
 if (NonDisplay) then
 :
 end if

Errors: E_NOFLDS Screen contains no fields or has
 not been scanned
 E_FLDNUM Invalid field number

20.4.5.5 GetCursorPosition

Syntax: **action GetCursorPosition**

Input None

Variables:

Output **Row** (integer)

Variables: **Column** (integer)

Remarks: **GetCursorPosition** returns the current 3270 screen cursor position by row and column.

Example: action GetCursorPosition
 if ((Row = 24) and (Column = 56)) then
 make Intro_Screen visible
 :
 end if

20.4.5.6 GetFieldNumber

Syntax: **action GetFieldNumber**

Input **Row** (integer)

Variables: **Column** (integer)

Output **FieldNumber** (integer)

Variables:

Remarks: **GetFieldNumber** finds the field number at the specified screen position. Screen positions start at (1,1) in the upper left corner of the screen.

If the given row and column are in the middle of a field, its field number is returned. If the given row and column are not inside a field, the number of the closest field further down the screen is returned. If there is an error, **FieldNumber** contains zero.

Example: # Determine which field contains the cursor
 action ScanScreen
 action GetCursorPosition # Returns row and column
 action GetFieldNumber # Uses row and column
 if (FieldNumber = 7) then
 :
 :
 end if

Errors: **E_NOFLDS** Screen contains no fields or has
 not been scanned
 E_ROWCOL (Row, Column) is not on screen

See Also: **GetCursorPosition**

20.4.5.7 GetFieldPosition

<i>Syntax:</i>	action GetFieldPosition	
<i>Input</i>	FieldNumber	(integer)
<i>Variables:</i>		
<i>Output</i>	Row	(integer)
<i>Variables:</i>	Column	(integer)
<i>Remarks:</i>	GetFieldPosition returns the starting row and column position of a specified field; (1,1) is the upper left corner of the screen buffer. If there is an error, (0,0) will be returned.	
<i>Example:</i>	copy 7 to FieldNumber action GetFieldPosition if ((Row = 2) and (Column = 28)) then : end if	
<i>Errors:</i>	E_NOFLDS	Screen contains no fields or has not been scanned
	E_FLDNUM	Invalid field number

20.4.5.8 ReadLine

Syntax: **action ReadLine**

Input **Row** (integer)
Variables:

Output **LineText** (string)
Variables:

Remarks: **ReadLine** copies the contents of line number **Row** of the screen buffer into the string variable **LineText**.

ReadLine is provided for use with screens that are line-oriented (unformatted). If it is applied to a screen that contains fields, the field information will be ignored.

Example: response to start
 copy "B" to SessionName
 action Init3270
 copy 1 to Row
 action ReadLine
 extract from LineText
 take word ScreenID
 if (ScreenID="MIN01001") then
 make Intro_Region visible
 end if

Errors: **E_LINENUM** Invalid line number

20.4.5.9 FindLastNonBlankLine

Syntax: **action FindLastNonBlankLine**
 Input **StartLine** (integer)
Variables:
 Output **Row** (integer)
Variables:

Remarks: **FindLastNonBlankLine** returns the number of the last non-blank line on the screen at or above **StartLine**. Zero is returned if all lines at or above **StartLine** are blank or if there is an error.

FindLastNonBlankLine is provided for use on screens that are line-oriented (unformatted), allowing you to locate the most recent prompt or message displayed by the application. If it is applied to a screen that contains fields, the field information will be ignored.

Example: action CheckOptions is
 copy 22 to StartLine # Search starts at line 22.
 action FindLastNonBlankLine
 action ReadLine
 switch LineText is
 case char "-More-" is
 make More_Key visible
 case char "-End of Data-" is
 make Another_Key visible
 default is
 make Another_Key visible
 end switch

Errors: **E_LINENUM** Invalid line number

See Also: **ReadLine**

20.4.5.10 ReadScreen

Syntax: **action ReadScreen**

Input **TopRow** (integer)
Variables: **BottomRow** (integer)
 LeftCol (integer)
 RightCol (integer)
 TextRegionName (string)

Output Text from the screen buffer is inserted into textual region
Variables: **TextRegionName.**

Remarks: **ReadScreen** inserts the contents of the 3270 screen buffer into a textual region. You can specify any rectangular area within the buffer by setting values for **TopRow**, **BottomRow**, **LeftCol**, and **RightCol**. The initial values of these variables are rows 1 through 24 and columns 1 through 80.

Note: If the screen size of the active connected session is other than Mod 2 (24 x 80) and the entire screen buffer is to be read, you must explicitly copy appropriate values to these variables; do not rely on the initial values.

You must clear the textual region prior to calling this action statement, unless you want the text inserted into the textual region at the current text cursor position.

Note: The contents of those fields with a non-display attribute on the 3270 terminal are visible in an EASEL OS/2 EE textual region.

Example: textual region Screen_1
 window size 80 columns 24 lines at position 100 100
 :
 clear Screen_1 # Clears textual region
 copy 5 to TopRow # Reads area of screen
 copy 20 to BottomRow # within rows 5-20 and
 copy 10 to LeftCol # columns 10-70.
 copy 70 to RightCol
 copy "Screen_1" to TextRegionName
 action ReadScreen

Errors: **E_SCRNAREA** Invalid screen area limits

See Also: **ReadLine**

20.4.5.11 ReadColoredScreen

Syntax: **action ReadColoredScreen**

Input **TopRow** (integer)

Variables: **BottomRow** (integer)

LeftCol (integer)

RightCol (integer)

TextRegionName (string)

Output Colored text is inserted into the colored textual region

Variables: **TextRegionName.**

Remarks: **ReadColoredScreen** inserts the contents of the 3270 screen buffer into a colored textual region. You can specify any rectangular area within the buffer by setting values for **TopRow**, **BottomRow**, **LeftCol**, and **RightCol**. The initial values of these variables are rows 1 through 24 and columns 1 through 80.

Note: If the screen size of the active connected session is other than Mod 2 (24 x 80) and the entire screen buffer is to be read, you must explicitly copy appropriate values to these variables; do not rely on the initial values.

You must clear the colored textual region prior to calling this action statement, unless you want the text inserted into the textual region at the current text cursor position.

Some 3270 applications make use of a special buffer which contains *extended field attributes* and *extended character attributes*. These extended attributes allow fields and individual characters within fields to be displayed with more colors than is normally possible. In the absence of extended attributes, four different colors are possible, and they are determined solely by field attributes:

Field Attribute		Color of Characters in Field	
<i>Protection</i>	<i>Intensity</i>	<i>Foreground</i>	<i>Background</i>
Unprotected	Normal	Green	Black
Unprotected	Intensified	Red	Black
Protected	Normal	Blue	Black
Protected	Intensified	White	Black

Note: The contents of those fields with a non-display attribute on the 3270 terminal are also not visible in an EASEL OS/2 EE **colored textual region**.

Errors: E_SCRNAREA Invalid screen area limits

See Also: **ReadColoredScreen**

20.4.6 Synchronizing EASEL OS/2 EE with the Host

Host transmissions to the screen buffer take time. If EASEL OS/2 EE examines the screen buffer too soon, it may attempt to perform actions based on an incomplete screen. Each 3270 screen has a “behavior pattern” that indicates the completion of its transmission. The action routines described in this section allow you to define and wait for these patterns.

EASEL OS/2 EE determines when the new screen is ready based on information you provide via the **M3270 watch** facility. This facility allows a program to detect and react to modifications made to the screen buffer during data transmission from the host application.

In order to use the **watch** facility, you first define a sequence of one or more **watch** commands. These define a set of conditions which must all be satisfied, *in sequence*, before the transmission of a screen is considered to be complete. You then invoke the appropriate action routine which tells 3270 support for EASEL OS/2 EE to start watching for those conditions to be met. For example, you can define a **watch** command sequence that expresses the following:

“Watch the cursor until it is not located at position row 24, column 6. Next, watch the Operator Information Area until no ‘X’ appears there. Let me know when this is done.”

(This is the example shown in the description of the **WatchAndWait** action routine.)

You can define **watch** commands to monitor any of the following:

1. The contents of a character position within the screen buffer
2. The cursor position
3. The “X” in the Operator Information Area

You can tailor the **watch** command sequences to the way your host operating system and application update the 3270 terminal screens. It will be easier for you to define appropriate **watch** command sequences by first answering the following questions on how the 3270 screens are updated:

1. When does the “X” in the Operator Information Area appear and disappear? Does it flicker as characters are written to the screen? Does it stay on continuously while waiting for the host to respond, or does it go on only when the host is transmitting?
2. Does the cursor position change from the previous screen? If so, when—after the “X” disappears, or just before?
3. Which characters will change from the previous screen? Which ones will stay the same? Will the screen be cleared before the next screen is displayed?

The answers to these questions will enable you to write EASEL OS/2 EE code that determines when the 3270 screen buffer has been completely updated.

You define a watch command sequence by first calling the **DefineWatch** action routine and then making one or more calls to the following action routines:

WatchForCursorAt	WatchForChar
WatchForCursorNotAt	WatchForNotChar
WatchForNoX	

Each of these actions encodes the requested watch command by appending some characters to the string variable **WatchCommandString**. These characters are interpreted by 3270 support for EASEL OS/2 EE at the time the watch commands are executed. (The contents of **WatchCommandString** are managed by 3270 support for EASEL OS/2 EE, so it is not necessary to know how the commands are encoded.)

The action routines **Watch** and **WatchAndWait** cause the current watch command sequence to be executed. Normally, one of these is called immediately after sending a keystroke that causes the host to transmit a new screen.

20.4.6.1 DefineWatch

Syntax: **action DefineWatch**

Input None

Variables:

Output **WatchCommandString** (string)

Variables:

Remarks: **DefineWatch** clears **WatchCommandString** in preparation for defining a new sequence of watch commands to be executed by the **Watch** or **WatchAndWait** action routines.

WatchCommandString is internal to the **M3270** module, and should not be referenced directly.

 If a single behavior pattern defines the completion of screen transmission for *all* screens in the 3270 application, **DefineWatch** need only be called once. If the screens do not behave in the same fashion, **DefineWatch** should be called each time a screen with changed behavior is expected.

Examples: *See Examples for:*

WatchForCursorAt
 WatchForCursorNotAt
 WatchForNotX
 WatchForChar
 WatchForNotChar

20.4.6.2 WatchForCursorAt

Syntax: **action WatchForCursorAt**

Input **WatchRow** (integer)

Variables: **WatchCol** (integer)

Output **WatchCommandString** (string)

Variables:

Remarks: **WatchForCursorAt** watches the 3270 screen until the cursor is located at the position specified by **WatchRow** and **WatchCol**.

WatchForCursorAt does not get executed until a call is made to the **Watch** or **WatchAndWait** action routine.

WatchCommandString is internal to the **M3270** module, and should not be referenced directly.

Example: action Sequence_1 is
 action DefineWatch
 copy 22 to WatchRow
 copy 54 to WatchCol
 action WatchForCursorAt

See Also: **WatchCursorNotAt**

20.4.6.3 WatchForCursorNotAt

Syntax: **action WatchForCursorNotAt**

Input **WatchRow** (integer)
Variables: **WatchCol** (integer)

Output **WatchCommandString** (string)
Variables:

Remarks: **WatchForCursorNotAt** watches the 3270 screen until the cursor is located at any position *other* than the one specified by **WatchRow** and **WatchCol**. In other words, it tests for the disappearance of the cursor from the specified position.

This command does not get executed until a call is made to the **Watch** or **WatchAndWait** action routine.

WatchCommandString is internal to the **M3270** module, and should not be referenced directly.

Example: action Sequence_2 is
 action DefineWatch
 copy 22 to WatchRow
 copy 54 to WatchCol
 action WatchForCursorNotAt

See Also: **WatchForCursorAt**

20.4.6.4 WatchForNoX

Syntax: **action WatchForNoX**

Input **SettleTime** (integer)

Variables:

Output **WatchCommandString** (string)

Variables:

Remarks: **WatchForNoX** watches the Operator Information Area until the “X” is gone and remains gone for the period of time specified by **SettleTime**.

SettleTime is in units of .01 seconds (the value of 100 is equivalent to 1 second). When the value of **SettleTime** is zero, which it is initially, **WatchForNoX** completes immediately once the “X” is gone. Specifying a non-zero value for **SettleTime** is useful for 3270 applications where the “X” flickers while the controller is writing data to the screen. Use of an appropriate value will ensure that the “X” is no longer flickering. If you do need to increase the value, it should be at least 20 (.20 seconds).

WatchForNoX does not get executed until a call is made to the **Watch** or **WatchAndWait** action routine.

WatchCommandString is internal to the **M3270** module, and should not be referenced directly.

Example: action Sequence_3 is
 action DefineWatch
 copy 20 to SettleTime
 action WatchForNoX
 copy 15 to WatchRow
 copy 56 to WatchCol
 action WatchForCursorAt

See Also: **GetXstatus**

20.4.6.5 WatchForChar

<i>Syntax:</i>	action WatchForChar	
<i>Input</i>	WatchRow	(integer)
<i>Variables:</i>	WatchCol	(integer)
	WatchChar	(string)
<i>Output</i>	WatchCommandString	(string)
<i>Variables:</i>		
<i>Remarks:</i>	WatchForChar watches the 3270 screen until the character specified by WatchChar appears at the location specified by WatchRow and WatchCol. In other words, WatchForChar tests for the appearance of a specific character at a specific location in the 3270 screen. If WatchChar contains more than one character, only the first character will be used. If you need to check for multiple characters on the 3270 screen, you can set up multiple WatchForChar definitions. WatchForChar does not get executed until a call is made to the Watch or WatchAndWait action routine. WatchCommandString is internal to the M3270 module, and should not be referenced directly.	
<i>Example:</i>	<pre>action Sequence_4 is action DefineWatch copy 22 to WatchRow copy 54 to WatchCol copy "Q" to WatchChar action WatchForChar</pre>	
<i>See Also:</i>	WatchForCursorNotAt	

20.4.6.6 WatchForNotChar

Syntax: **action WatchForNotChar**

Input **WatchRow** (integer)

Variables: **WatchCol** (integer)

WatchChar (string)

Output **WatchCommandString** (string)

Variables:

Remarks: **WatchForNotChar** watches the 3270 screen until any character other than the one specified by **WatchChar** appears at the location specified by **WatchRow** and **WatchCol**. In other words, **WatchForNotChar** tests for the disappearance of a specific character at a specific location in the 3270 screen.

If **WatchChar** contains more than one character, only the first character will be used. If you need to check for multiple characters that are not on the 3270 screen, you can set up multiple **WatchForNotChar** definitions.

WatchForNotChar does not get executed until a call is made to the **Watch** or **WatchAndWait** action routine.

WatchCommandString is internal to the **M3270** module, and should not be referenced directly.

Example: action Sequence_5 is
 action DefineWatch
 copy 22 to WatchRow
 copy 54 to WatchCol
 copy "V" to WatchChar
 action WatchForNotChar

See Also: **WatchForCursorAt**

20.4.6.7 Watch

<i>Syntax:</i>	action Watch	
<i>Input</i>	WatchCommandString	(string)
<i>Variables:</i>	GiveUpTime	(integer)
<i>Output</i>	None	
<i>Variables:</i>		
<i>Remarks:</i>	Watch executes the watch commands in WatchCommandString while allowing EASEL OS/2 EE to continue to respond to other stimuli that do not pertain to the 3270 buffer (such as object selections and keyboard input), as long as those responses are within a guarded or resumable begin/end block. The watch commands are executed one at a time, in the same order in which they were defined (via the WatchFor... action routines), until either there are no more watch commands left, or a period of time specified by GiveUpTime has elapsed since Watch was called. GiveUpTime is in units of .01 seconds (the value 100 is equivalent to 1 second). Its initial value is 3000 (30 seconds). To detect the completion of Watch, you must set up a guarded or resumable begin/end block with <i>both</i> of the following response to line statements, exactly as shown: response to line "Watch done" from A3270 <i>and:</i> response to line "Watch gave up" from A3270 "Watch done" and "Watch gave up" are status messages from the A3270 local application. Normally, the message "Watch done" will be received, unless GiveUpTime elapses before all of the watch commands complete or an error has occurred.	

It is highly recommended to define the block as either **guarded** or **resumable**. If you want to restrict valid responses to be only those within the block, make the block **guarded**. If you want to have all other responses in outer blocks valid as well, make the block **resumable**. The begin/end block must have at least one **leave block** statement.

The **Watch** action routine returns immediately after being called, but the watch commands continue executing until one of the above status messages is received from A3270. Calling any other **M3270** action routine while a watch is in progress will interrupt and cancel the watch. No status message will result from a canceled watch.

Example:

```

action DefineWatch
copy 24 to WatchRow      # The developer knows that the
copy 79 to WatchCol      # next screen has an asterisk at
copy "*" to WatchChar    # row 24, column 79.
action WatchForChar
action PressENTER        # Pressing the ENTER key causes
                        # the next screen to be displayed
                        # in this example.
# Watch for the next screen to be completed:
copy 2000 to GiveUpTime  # Give up after 20 seconds.
action Watch
begin guarded
    response to line "Watch done" from A3270
        copy "Mo_Report" to TextRegionName
        action ReadScreen
        leave block

    # If the watch did not complete within 20 seconds,
    # have user try again.
    response to line "Watch gave up" from A3270
        change Display_Message to
        text "System error, try again"
        make Display_Message visible
        leave block

    response to Menu_keys
    # Respond to keys while
    # waiting for the next screen. All other active
    # responses must be inside this block.
:
end

```

Errors: **E_WATCH** Empty or invalid watch command string

See Also: **WatchandWait**

20.4.6.8 WatchAndWait

Syntax: **action WatchAndWait**

Input **WatchCommandString** (string)

Variables: **GiveUpTime** (integer)

Output **WatchGaveUp** (boolean)

Variables:

Remarks: **WatchAndWait** executes the watch commands in **WatchCommandString** in the same way as **Watch**, except that it does *not* allow EASEL OS/2 EE to respond to other stimuli while **WatchAndWait** is running. Therefore, no begin/end blocks needs to be defined.

The watch commands are executed one at a time, in the same order in which they were defined (via the **WatchFor...** action routines), until either there are no more watch commands left, or a period of time specified by **GiveUpTime** has elapsed since **WatchAndWait** was called.

GiveUpTime is in units of .01 seconds (the value 100 is equivalent to 1 second). Its initial value is 3000 (30 seconds).

WatchGaveUp is set to either **true** or **false**, depending on whether or not the **GiveUpTime** has elapsed.

Example:

```
action DefineWatch                    # The developer knows that
copy 22 to WatchRow                  # the next screen will be
copy 6 to WatchCol                   # complete when the cursor
action WatchForCursorNotAt           # disappears from row 22,
copy 0 to SettleTime                  # column 6.
action WatchForNoX
action PressENTER                    # Pressing the ENTER key will cause
                                      # the next screen to be displayed
                                      # in this example.
copy 2500 to GiveUpTime               # Wait for the next screen
action WatchAndWait                  # to be completed. If not
if (WatchGaveUp) then                # completed in 25 seconds,
    change Display_Message to        # have user try again.
    text "System error, try again"
    make Display_Message visible
else
    copy "Weekly_report" to TextRegionName # If completed
    action ReadScreen                        # within 25
end if                                        # seconds, use
                                             # it.
```

Errors: **E_WATCH** Empty or invalid watch command
 string

See Also: **Watch**

20.4.7 Communicating with the Host

Host applications may require user responses, including keyed data, menu selections, and special function keys. These are provided through action routines which simulate various keyboard functions.

20.4.7.1 PressKey

Syntax: **action PressKey**

 Input **KeyCode** (integer)

Variables: **AutoWait** (boolean)

 Output None

Variables:

Remarks: **PressKey** simulates the pressing of a non-printing key that has special significance, such as “tab,” “enter,” “clear,” etc. Available key codes are shown in the table below; copy one of these integer constants to **KeyCode**.

ATTN	DOWN_ARROW	IDENT	PRINT
BACK_TAB	DUP	INSERT	RESET
CLEAR	ENTER	LEFT_ARROW	RIGHT_ARROW
CURSR_SEL	ERASE_EOF	NEW_LINE	SYS_REQ
DBL_LEFT_ARROW	ERASE_INPUT	PA1	TAB
DBL_RIGHT_ARROW	FIELD_MARK	PA2	TEST
DELETE	HOME	PA3	UP_ARROW
DEV_CNCL			

 If **AutoWait** is set to “true,” the routine calls **WatchAndWait** before returning, to provide automatic synchronization.

Example: response to NextField
 copy TAB to KeyCode
 action PressKey

Errors: **E_SENDKEY** Failed to send keystroke(s)

See Also: **PressENTER**
 PressPFkey
 TypeString
 EnterString
 WatchandWait

20.4.7.2 PressENTER

<i>Syntax:</i>	action PressENTER	
<i>Input</i>	AutoWait	(boolean)
<i>Variables:</i>		
<i>Output</i>	None	
<i>Variables:</i>		
<i>Remarks:</i>	PressENTER simulates the pressing of the ENTER key on a 3270 terminal. Alternatively, this operation can be performed by calling PressKey with KeyCode set to ENTER. However, PressENTER is provided for convenience, since the ENTER key is used frequently for many applications. It does not alter the value of KeyCode. If AutoWait is set true, the routine calls WatchAndWait before returning.	
<i>Example:</i>	<pre>response to RptSelect copy true to AutoWait action PressENTER if (WatchGaveUp) then : else : end if</pre>	
<i>Errors:</i>	E_SENDKEY	Failed to send keystroke(s)
<i>See Also:</i>	PressKey PressPFkey TypeString EnterString WatchandWait	

20.4.7.3 PressPFkey

Syntax: **action PressPFkey**

Input **AutoWait** (boolean)

Variables: **PFkeyNumber** (integer)

Output None

Variables:

Remarks: **PressPFkey** simulates the pressing of the PF key specified by **PFkeyNumber** on a 3270 terminal—in other words, one of the special function keys on the terminal keyboard.

PFkeyNumber must be a number from 1 to 24.

 If **AutoWait** is set true, the routine calls **WatchAndWait** before returning.

Example: response to Another_Key
 copy 5 to PFkeyNumber # Press PF5
 action PressPFkey

 response to More_Key
 copy 1 to PFkeyNumber # Press PF1
 action PressPFkey

Errors: **E_SENDKEY** Failed to send keystroke(s)

See Also: **PressKey**
 PressENTER
 TypeString
 EnterString
 WatchAndWait

20.4.7.4 TypeString

<i>Syntax:</i>	action TypeString
<i>Input</i>	Keystrokes (string)
<i>Variables:</i>	
<i>Output</i>	None
<i>Variables:</i>	
<i>Remarks:</i>	TypeString simulates the typing of a string of data on a 3270 terminal keyboard. The string of data is specified by the variable Keystrokes. Keystrokes can contain only printing ASCII characters, including spaces and tabs (an ASCII tab character, "\t," translates to pressing the TAB key on a 3270 terminal). Keystrokes can also contain characters from the IBM PC Extended Character Set. Characters in the string are entered into the screen buffer starting at the cursor location, which is usually at the beginning of the field. If the string length equals or exceeds the field size, the consequences are identical to what would happen if the user entered the string directly from a 3270 terminal keyboard, and are dependent upon the 3270 application.
<i>Example:</i>	<pre>enabled blue region Manufacturing size 400 40 at position 110 90 in Screen_1 white border parameter is "MF2599" move to 200 20 text center "MANUFACTURING" class RptSelect is Stat Defect Warehouse Manufacturing # MF2599 is typed in at the cursor position in the # screen buffer, followed by the ENTER key. response to RptSelect copy parameter of object to Keystrokes action TypeString action PressENTER</pre>
<i>Errors:</i>	E_SENDKEY Failed to send keystroke(s)
<i>See Also:</i>	PressKey PressENTER TypeString EnterString

20.4.7.5 EnterString

<i>Syntax:</i>	action EnterString	
<i>Input</i>	Keystrokes	(string)
<i>Variables:</i>	AutoWait	(boolean)
<i>Output</i>	None	
<i>Variables:</i>		
<i>Remarks:</i>	EnterString is similar to TypeString except that the additional keystroke, ENTER, is sent. This is slightly faster than using TypeString followed by PressENTER . If AutoWait is set to "true," the routine calls WatchAndWait before returning.	
<i>Example:</i>	response to RptSelect copy parameter of object to Keystrokes action EnterString	
<i>Errors:</i>	E_SENDKEY	Failed to send keystroke(s)
<i>See Also:</i>	PressKey PressPFkey PressENTER TypeString WatchandWait	

20.4.8 Controlling the Emulator Window

20.4.8.1 SetEmulatorWindow

Syntax: **action SetEmulatorWindow**
Input **EmulatorWindowCommand** (integer)
Variables:

Output None
Variables:

Remarks: **SetEmulatorWindow** is used for changing the state of the 3270 emulator window associated with the current session. The emulator window can be made visible or invisible, be activated, deactivated, maximized, minimized, or restored (i.e., changed to its normal window size).

The value of the input variable **EmulatorWindowCommand** determines the new state of the emulator window. The following integer constants must be used for copying a value to **EmulatorWindowCommand**:

Integer Constant	Effect
EW_MAKE_VISIBLE	Makes the emulator window visible.
EW_MAKE_INVISIBLE	Makes the emulator window invisible.
EW_ACTIVATE	Activates the emulator window (sets focus to the window and places it in the foreground).
EW_DEACTIVATE	Deactivates the emulator window (removes input focus and places the window behind all other windows).
EW_MINIMIZE	Minimizes the emulator window.
EW_MAXIMIZE	Maximizes the emulator window.
EW_RESTORE	Restores the emulator window to its normal appearance (neither minimized nor maximized).

These values can be combined by adding two or more together as long as they do not conflict (see example).

Actions **Connect** and **Disconnect** do not affect the appearance of the emulator window. However, you should consider using **SetEmulatorWindow** to make the emulator window invisible after connecting to a session for the first time and to make it visible just before disconnecting from the session. This ensures the user will not inadvertently interfere with EASEL OS/2 EE 3270 communications by activating the emulator and typing keystrokes. (Actions **Emulate3270** and **EmulateAndWatch**, also described in this section, are useful when you do want the user to use the emulator.)

Example: # Activate and maximize the emulator window.
 copy (EW_ACTIVATE + EW_MAXIMIZE) to EmulatorWindowCommand
 action SetEmulatorWindow

20.4.8.2 Emulate3270

Syntax: **action Emulate3270**

Input None

Variables:

Output None

Variables:

Remarks: **Emulate3270** places the user in the windowed emulator by making it visible, active, and maximized. It does not return until the user minimizes the emulator window. Upon returning, the emulator window is returned to the appearance it held previous to calling this action. (Therefore, the emulator window only remains minimized if it had been minimized at the time **Emulate3270** was called.)

Emulate3270 does not affect the appearance of windows belonging to the EASEL OS/2 EE program. We recommend that the primary window of the program be made invisible before calling **Emulate3270** and be made visible once it returns (see example).

Example: # Make EASEL OS/2 EE program windows invisible
 # while the emulator window is active.
 make PrimaryRegion invisible
 action Emulate3270
 make PrimaryRegion visible

20.4.8.3 EmulateAndWatch

<i>Syntax:</i>	action EmulateAndWatch
<i>Input</i>	ScreenIDFile (string)
<i>Variables:</i>	
<i>Output</i>	ScreenName (string)
<i>Variables:</i>	

Remarks: **EmulateAndWatch** is similar to action **Emulate3270** in that it places the user in the windowed emulator by making it visible, active, and maximized. The difference is that **EmulateAndWatch** allows you to specify one or more screens which, when encountered by the emulator, automatically cause the action to return.

EmulateAndWatch allows you to effectively combine an EASEL OS/2 EE front-end with terminal emulation and switch between these. Using it, you can implement an EASEL OS/2 EE interface to a host application in an incremental fashion; those screens for which you have not yet written a graphical interface can be accessed by the user via the emulator. Unlike action **Emulate3270**, **EmulateAndWatch** lets you restrict the user to accessing only a particular set of screens while in the emulator. As soon as the emulator encounters a screen which your program knows about, **EmulateAndWatch** returns and EASEL OS/2 EE regains control of the host application.

You define which screens cause EASEL OS/2 EE to regain control by creating a screen ID file. You copy the name of this file to the variable **ScreenIDFile** before calling **EmulateAndWatch**. A screen ID file must contain one or more sets of lines in the following format:

```
screen SCREEN_NAME
  ROW_NUM1 COL_NUM1 MATCH_STRING1
  ROW_NUM2 COL_NUM2 MATCH_STRING2
  . . . . .
```

Each set of lines defines a screen that causes EASEL OS/2 EE to regain control from the emulator. **SCREEN_NAME** is any name or word you want to use to identify a screen. When **EmulateAndWatch** returns, this name appears in the output variable **ScreenName**. When more than one screen is specified in the screen ID file, **ScreenName** allows you to easily determine which of the screens has caused **EmulateAndWatch** to return and is, therefore, currently being displayed by the host application. If this action returns because the user has minimized the emulator window, **ScreenName** contains an empty string.

Each line beginning with **screen** must be followed by one or more lines containing a match string. These lines specify how the screen must appear. The screen is not recognized unless the string **MATCH_STRING** appears in the screen buffer, exactly as specified, starting at the location specified by **ROW_NUM** and **COL_NUM**. If multiple match-string lines are listed below a **screen** line, all of them must be satisfied in order for that screen to be recognized. **MATCH_STRING** must be delimited by double-quotation marks (as are EASEL OS/2 EE string literals). A double-quotation mark can appear inside **MATCH_STRING** if it is preceded by the EASEL OS/2 EE escape character "\". The **MATCH_STRING** should not contain any null characters. The **ROW_NUM**, **COL_NUM**, and **MATCH_STRING** must all be specified and listed on one line.

Upon returning, the emulator window is returned to the appearance it held previous to calling this action. **EmulateAndWatch** does not affect the appearance of windows belonging to the EASEL OS/2 EE program; we recommend that the primary window of the program be made invisible before calling **EmulateAndWatch** and be made visible once it returns (see example).

Example: File screenid.dat:

```
screen MainMenu1
  3 39 "MAIN MENU"
  3 65 "Page 1 of 2"
screen MainMenu2
  3 39 "MAIN MENU"
  3 65 "Page 2 of 2"
screen ExitScreen
  5 20 "Type \"x\" to exit application"
```

EASEL OS/2 EE code:

```
action MainMenu1 is
:
action MainMenu2 is
:
action ExitScreen is
:
action UnknownScreen is
:
copy "screenid.dat" to ScreenIDFile
make PrimaryRegion invisible
action EmulateAndWatch
make PrimaryRegion visible
if (ScreenName != "") then
  # Emulator encountered a known screen
  action ScreenName
else
  # User minimized emulator window
  action UnknownScreen
end if
```

<i>Errors:</i>	E_SCRIDFILE	Screen ID file not found
	E_BADSCRID	Screen ID file has illegal syntax
	E_NOMEMORY	Not enough memory for screen IDs

20.4.9 Miscellaneous Action Routines

20.4.9.1 WriteField

Syntax: **action WriteField**

Input **FieldNumber** (integer)

Variables: **FieldText** (string)

FillFlag (boolean)

Output None

Variables:

Remarks: **WriteField** simulates the typing of a string of data on a 3270 terminal keyboard into a field specified by the variable **FieldNumber**.

If the field specified by **FieldNumber** is not protected, it is written with the string contained in the variable **FieldText**.

If the field is longer than **FieldText** and if **FillFlag** is set to "true," then the field is padded on the right with null characters (blanks). If **FillFlag** is set to "false," the original field characters that were not overwritten by the input string will remain in the field.

If **FieldText** is longer than the field, trailing characters of **FieldText** will be truncated.

Example: response to RptSelect # Assumes
 copy 7 to FieldNumber # that screen
 copy true to FillFlag # has been
 copy parameter of object to FieldText # previously
 action WriteField # scanned by
 action PressENTER # another
 # response.

Errors: **E_NOFLDS** Screen contains no fields or has
 not been scanned

E_FLDNUM Invalid field number

E_PROTFLD Attempt to write protected field

20.4.9.2 GetXstatus

Syntax: **action GetXstatus**

Input None

Variables:

Output **Xstatus** (integer)

Variables:

Remarks: **GetXstatus** queries the status of the “X” in the Operator Information Area (the condition causing the “X” to appear in the Operator Information Area). The condition is returned in the variable **Xstatus**.

The constants shown in the table below correspond to messages the host displays on the Operator Information Area. You should use these to test the value of **Xstatus**.

Example: response to line "Watch gave up" from A3270

```
          action GetXstatus
          if (Xstatus = Xprtbsy) then
            change Display_Message to
              text "Printer is busy"
            make Display_Message visible
          end if
```

Constant Name	Message	Description/Action
Xready	-	No “X” is present.
Xclock	Time	Time is required for the host system to perform a function. Wait.
Xsystem	System Lock	The host system has locked your keyboard. Look for a message. Wait or Press RESET.
Xmechck	Machine Check	Mechanical malfunction. The display station is not working properly.
Xcommck	Communication Check	There is a problem with the communication line between the control unit and the host system. Press RESET.
Xprogck	Program Check	The control unit detected a programming error in the data it received from the host system. Press RESET.

Constant Name	Message	Description/Action
Xwhat	What?	Last input symbol not accepted. Try again.
Xnofun	Minus Function	Requested function not available. Press RESET.
Xbadfun	Minus Function Operator	User not authorized for requested function. Press RESET.
Xkey	Security Key	Security keylock is off. No operator input can be accepted.
Xnoprt	Printer Not Working	Printer not working/available. This may appear for a host-initiated print operation.
Xprtbsy Xprtvr Xunauth	Printer Busy Printer Very Busy Operator Not Authorized	Your display station is not authorized to do the requested printer-related function. Press RESET.
Xgoaway	Go Elsewhere	You attempted to take an action in the wrong location. Press RESET. Move the cursor or take other action.
Xmore	Too Much Data	You have tried to insert more data than this field can hold. Press RESET. Correct the entry.
Xnum	Numeric Data Only	Attempt to enter non-numeric data in numeric field. Press RESET. Then enter only numerals.
Xwhtnum	What Number?	The number entered is out of range or invalid. Press RESET and make correct entry.
Xbadcrd	Questionable Card	Bad or damaged mag stripe on card. The RESET key should be pressed and a good card used.
Xnosym	- Minus Symbol	The symbol you pressed is not available. Press RESET to restore keyboard.
Xunknown		X is displayed but the specific Operator Information Area condition cannot be deciphered.

For further information, see the *IBM 3270 Information Display System 3278 Display Station Operator's Guide*.

See Also: **WatchForNoX**

20.4.9.3 Print3270Screen

Syntax: **action Print3270Screen**

Input None

Variables:

Output None

Variables:

Remarks: **Print3270Screen** prints the contents of the current 3270 screen buffer. This routine is useful for developing and debugging EASEL OS/2 EE programs. Do not call this action unless your printer is connected to LPT1.

Example: response to line "print" from keyboard
 action Print3270Screen

20.4.9.4 GetSessionStatus

Syntax: **action GetSessionStatus**

 Input None

Variables:

 Output **SessionStatus** (integer)

Variables:

Remarks: **GetSessionStatus** returns the readiness status and system connection information of the current session in the output variable **SessionStatus**. This status value is derived from the symbols displayed in the leftmost portion of the Operator Information Area. Use the following constants for testing the value of **SessionStatus**:

Constant Name	3278/79 Symbol	Description
SS_DISCONNECTED		The session is in disconnected mode. Usually this means that there is no physical connection to the host; for example, the coax cable is not attached to the workstation, or the modem is not dialed up.
SS_CONTROL		The session is connected and the display is owned by the system operator (also known as "SSCP Control Program"). This is often the status when the session is not logged on and the logon screen is being displayed.
SS_APPLICATION		The session is connected and the display is owned by an application program.
SS_UNOWNED		The session is connected, but the display is not owned by the system operator or by an application program.

The symbols shown here are those displayed in the third column position in the Operator Information Area on a 3278/79 terminal. Symbols used for indicating readiness and system connect vary among workstation terminal emulators; refer to your emulator's documentation to identify the symbols used by your emulator.

Example: response to start
 copy "A" to SessionName
 action Init3270
 action GetSessionStatus
 if (SessionStatus = SS_DISCONNECTED or
 SessionStatus = SS_UNOWNED) then
 change Message to text "Cannot log in to host"
 make Message visible
 wait 4
 action Stop3270
 exit
 else if (SessionStatus = SS_CONTROL) then
 action Login_Sequence
 end if
 end if

See Also: **GetXstatus**

20.4.9.5 GetSessionInfo

Syntax: **action GetSessionInfo**

Input **SessionName** (string)

Variables:

Output **SessionType** (integer)

Variables: **NumRows** (integer)

NumCols (integer)

Remarks: Action **GetSessionInfo** returns information about the session specified by **SessionName**. **SessionName** can contain the short session name of any started session, whether or not it is currently connected. When **GetSessionInfo** returns, **SessionType** contains a value representing the type of session and **NumRows** and **NumCols** contain the dimensions of the session's screen buffer. (See 20.2.2, Screen Sizes, which lists Model numbers and their dimensions.)

Note that the value returned in **M52_NumRows** does not include the Operator Information Area. Also, the values returned in **M52_NumRows** and **M52_NumCols** are not meaningful for printer sessions.

The following integer constants must be used when testing the value of **SessionType**:

Integer Constants	Meaning
ST_3270_HOST	3270 host terminal session
ST_3270_PRINTER	3270 printer session
ST_5250_HOST	5250 host terminal session
ST_5250_PRINTER	5250 printer session
ST_PC	Session is local to the Program- mable Workstation
ST_UNKNOWN	Session type cannot be deter- mined

The following values are returned if an error (**E_SESSNAME**) occurs:

Variable	Value
SessionType	ST_UNKNOWN
NumRows	0
NumCols	0

Example:

```

copy "D" to SessionName
action GetSessionInfo
If (SessionType = ST_3270_HOST) then
    action Connect
    if (not A3270Error) then
        # Read the entire screen buffer of the active
        # connected session into a textual region.
        copy 1 to TopRow
        copy NumRows to BottomRow
        copy 1 to LeftCol
        copy Numcols to RightCol
        copy "CurrentScreen" to TextRegionName
        clear CurrentScreen
        action ReadColoredScreen
    end if
end if

```

Errors: E_SESSNAME Invalid session name

20.5 Error Handling

20.5.1 Error Status Variables

The **M3270** module returns errors status information in three variables:

A3270Error	(boolean)
A3270ErrorCode	(integer)
A3270ErrorText	(string)

These variables are set each time an **M3270** action routine is called.

A3270Error is set to "true" if an error occurred during execution of the last action routine. Otherwise, it is set to "false."

When **A3270Error** is "true," the other two variables can be used to obtain more information about the error condition:

A3270ErrorCode	Contains an error code number that denotes the particular error that occurred.
A3270ErrorText	Contains an error message that also appears in the EASEL OS/2 EE errorlog file. If A3270Error is false, A3270ErrorCode contains zero and A3270ErrorText contains a null string.

Each action routine described in this manual lists the error conditions that can result when the action routine is called, showing the error code and the associated error message text. The error code value is always represented by an integer constant whose name begins with "E_" (see Reserved Constants listing). Always use these constants when testing the value of **A3270ErrorCode**.

In general, it is not necessary to have statements in your program which check for all possible errors after every action routine call. Many of the error conditions are caused by coding mistakes (as in the example below) and should never occur in a fully debugged program.

Example: The following EASEL OS/2 EE statements are executed:

```
copy -3 to Row    # Row should not be less
                  # than 1
action ReadLine   # ...this will cause an
                  # error
```


Errors: After **ReadLine** returns, the error variables will be set as follows:

A3270Error	= true
A3270ErrorCode	= E_LINENUM
A3270ErrorText	= "Invalid line number"

Where: E_LINENUM is an integer constant whose value is the error code corresponding to this error condition. The errorlog will contain the line:

3270 Error: Invalid line number

20.5.2 Error Messages (Cross Reference)

In the following, the error message in the EASEL OS/2 EE errorlog file is listed along with the constant that corresponds to this error condition, and a brief explanation.

20.5.2.1 Syntax Error in Command Line Sent from EASEL OS/2 EE

Message: **3270 Error: Syntax error in command line sent from EASEL OS/2 EE**

Constant: **E_COMMAND**

Remarks: A command was sent from the **M3270** module that could not be deciphered by the **A3270** local application. This error will never occur unless there is a version mismatch between **M3270** and **A3270**. Make sure EASEL OS/2 EE has been properly installed and no out-of-date versions are on your system.

20.5.2.2 Invalid Field Number

Message: **3270 Error: Invalid field number**

Constant: **E_FLDNUM**

Remarks: The value of **FieldNumber** was not a number between 1 and **FieldCount** inclusive.

See: **GetField, WriteField, GetFieldLength, GetFieldAttributes, GetFieldPosition**

20.5.2.3 Attempt to Write Protected Field

Message: **3270 Error: Attempt to write protected field**

Constant: **E_PROTFLD**

Remarks: Action **WriteField** was called with a protected field specified.

See: **WriteField**

20.5.2.4 Invalid Line Number

Message: **3270 Error: Invalid line number**

Constant: **E_LINENUM**

Remarks: The specified line number was not a value between 1 and n inclusive, where n is the number of lines contained in the screen buffer.

See: **ReadLine, FindLastNonBlankLine**

20.5.2.5 Invalid Screen Area Limits

Message: **3270 Error: Invalid screen area limits**

Constant: **E_SCRNAREA**

Remarks: The values of **TopRow**, **BottomRow**, **LeftCol**, and **RightCol** did not specify an area which resides entirely within the screen buffer.

See: **ReadScreen, ReadColoredScreen**

20.5.2.6 (Row, Column) Is Not on Screen

Message: **3270 Error: (Row, Column) is not on screen**

Constant: **E_ROWCOL**

Remarks: The input variables **Row** and **Column** specified a character position that was not within the screen buffer.

See: **ReadFieldNumber**

20.5.2.7 Failed to Send Keystroke(s)

Message: **3270 Error: Failed to send keystroke(s)**

Constant: **E_SENDKEY**

Remarks: An attempt to send a keystroke to the controller failed.

See: **PressKey, PressENTER, PressPFkey, TypeString, EnterString**

20.5.2.8 Not Initialized - Command Was Ignored

Message: **3270 Error: Not initialized - command was ignored**

Constant: **E_NOTINIT**

Remarks: **Init3270** was not the first **M3270** action routine called. All other action routines cause this error message until **Init3270** gets called.

20.5.2.9 Screen Contains No Fields or Has Not Been Scanned

Message: **3270 Error: Screen contains no fields or has not been scanned**

Constant: **E_NOFLDS**

Remarks: An action routine that uses **FieldNumber** was called when the value of **FieldCount** was zero.

See: **ReadField, WriteField, GetFieldLength, GetFieldAttributes, GetFieldNumber, GetFieldPosition**

20.5.2.10 Empty or Invalid WatchCommandString

Message: **3270 Error: Empty or invalid WatchCommandString**

Constant: **E_WATCH**

Remarks: No **WatchFor...** action routines were called prior to calling **Watch** or **WatchAndWait**.

See: **Watch, WatchAndWait**

20.5.2.11 Cannot Initialize

Message: **3270 Error: Cannot initialize**

Constant: **E_CANTINIT**

Remarks: **Init3270** could not initialize. This error indicates that required hardware or software is not installed, or Communications Manager (CM) has not been started.

See: **Init3270**

20.5.2.12 Not Connected to an Active Session

Message: **3270 Error: Not connected to an active session**
Constant: **E_NOTCONN**
Remarks: An **M3270** action routine was called which required an active session and there was none.

20.5.2.13 Invalid Session Name

Message: **3270 Error: Invalid session name**
Constant: **E_SESSNAME**
Remarks: The value of **SessionName** was not a valid session name. A valid name must be one of the 3270 host session short names configured in CM and it must have been started by CM.

20.5.2.14 Session Is Already in Use:

Message: **3270 Error: Session is already in use**
Constant: **E_INUSE**
Remarks: The name specified by **SessionName** was valid but the session was connected by another process.

20.5.2.15 Action Is Not Available in This Configuration

Message: **3270 Error: Action is not available in this configuration.**
Constant: **E_NOTAVAIL**
Remarks: This error should occur only if you are porting EASEL OS/2 EE interface originally developed for other than EASEL OS/2 EE. Be sure only those 3270 Support action routines described in 20.4, "M3270 Module Action Routines" on page 20-13 are being used.

20.5.2.16 Screen ID File Not Found

Message: **3270 Error: Screen ID file not found**
Constant: **E_SCRIDFILE**
Remarks: **EmulateAndWatch** could not find the file specified by **ScreenIDFile**.

20.5.2.17 Screen ID file has illegal syntax

Message: **3270 Error: Screen ID file has illegal syntax**

Constant: **E_BADSCRID**

Remarks: **EmulateAndWatch** detected a syntax error in the file specified by **ScreenIDFile**.

20.5.2.18 Not Enough Memory for Screen IDs

Message: **3270 Error: Not enough memory for screen IDs**

Constant: **E_NOMEMORY**

Remarks: **EmulateAndWatch** could not allocate enough memory for reading the file specified by **ScreenIDFile**.

20.6 Reserved Variables (Cross Reference)

The following reserved variables are used in the **M3270** module. They are all global variables. Be sure that your program does not contain any of these identifiers except when referencing the **M3270** module.

The initial values are set by the action routine **Init3270**.

Global Variable Name	Type	Initial Value	Action Routine References (O = output, I = input)
A3270Error	boolean	false	(all action routines)
A3270ErrorCode	integer	0	(all action routines)
A3270ErrorText	string	" "	(all action routines)
Alphanumeric	boolean	false	GetFieldAttributes (I)
AutoWait	boolean	false	PressKey (I) PressENTER (I) PressPFkey (I) EnterString (I)
BottomRow	integer	24	ReadScreen (I)
Column	integer	1	GetCursorPosition (O) GetFieldPosition (O) GetFieldNumber (I)
EmulatorWindowCommand	integer	EW_MAKE_INVISIBLE	SetEmulatorWindow (I)
FieldCount	integer	0	ScanScreen (O)
FieldLength	integer	0	GetFieldLength (O)
FieldNumber	integer	0	GetFieldNumber (O) ReadField (I) ReadLine (I) WriteField (I) GetFieldLength (I) GetFieldAttributes (I) GetFieldPosition (I)
FieldText	string	" "	ReadField (O) WriteField (I)
FillFlag	boolean	false	WriteField (I)
GiveUpTime	integer	3000	Watch (I) WatchAndWait (I)
Intensified	boolean	false	GetFieldAttributes (O)
KeyCode	integer	TAB	PressKey (I)
Keystrokes	string	" "	TypeString (I) EnterString (I)

Global Variable Name	Type	Initial Value	Action Routine References (O = output, I = input)
LeftCol	integer	1	ReadScreen (I)
LineText	string	" "	ReadLine (O)
Modified	boolean	false	GetFieldAttributes (O)
NonDisplay	boolean	false	GetFieldAttributes (O)
NumCols	integer	0	GetSessionInfo (O)
NumRows	integer	0	GetSessionInfo (O)
PFkeyNumber	integer	1	PressPFkey (I)
Protected	boolean	false	GetFieldAttributes (O)
RightCol	integer	80	ReadScreen (I)
Row	integer	1	GetCursorPosition (O) GetFieldPosition (O) FindLastNonBlankLine (O) ReadLine (I) GetFieldNumber (I)
ScreenIDFile	string	"screenid.dat"	EmulateAndWatch (I)
ScreenName	string	" "	EmulateAndWatch (O)
SessionName	string	"*"	Init3270 (I) Connect (I) GetSessionInfo (O)
SessionStatus	integer	SS_DISCONNECTED	GetSessionStatus (O)
SessionType	integer	ST_UNKNOWN	GetSessionInfo (O)
SettleTime	integer	0	WatchForNoX (I)
StartLine	integer	24	FindLastNonBlankLine (I)
TextRegionName	string	"TR3270"	ReadScreen (I)
TopRow	integer	1	ReadScreen (I)
WatchChar	string	" "	WatchForChar (I) WatchForNotChar (I)
WatchCol	integer	1	WatchForCursorAt (I) WatchForCursorNotAt (I) WatchForChar (I) WatchForNotChar (I)
WatchCommandString	string	" "	DefineWatch (O) WatchForCursorAt (O) WatchForCursorNotAt (O) WatchForChar (O) WatchForNotChar (O) WatchForNoX (O) Watch (I) WatchandWait (I)

Global Variable Name	Type	Initial Value	Action Routine References (O = output, I = input)
WatchGaveUp	boolean	false	WatchAndWait (O)
WatchRow	integer	1	WatchForCursorAt (I) WatchForCursorNotAt (I) WatchForChar (I) WatchForNotChar (I)
Xstatus	integer	Xready	GetXstatus (O)

20.7 Reserved Constants

The following reserved constants are used in the **M3270** module. Be sure that your program does not contain any of these identifiers except when referencing the **M3270** module.

ATTN	E_NOMEMORY	ST_5250_PRINTER
BACK_TAB	E_NOTINIT	ST_PC
CLEAR	E_PROTFLD	ST_UNKNOWN
CURSR_SEL	E_ROW_COL	SYS_REQ
DBL_LEFT_ARROW	E_SCRNAREA	TAB
DBL_RIGHT_ARROW	E_SCRIDFILE	TEST
DELETE	E_SENDKEY	UP_ARROW
DEV_CNCL	E_SESSNAME	Xbadcrd
DOWN_ARROW	E_WATCH	Xbadfun
DUP	FIELD_MARK	Xclock
ENTER	HOME	Xcommck
ERASE_EOF	IDENT	Xgoaway
ERASE_INPUT	INSERT	Xkey
EW_ACTIVATE	LEFT_ARROW	Xmechck
EW_DEACTIVATE	NEW_LINE	Xmore
EW_MAKE_INVISIBLE	PA1	Xnofun
EW_MAKE_VISIBLE	PA2	Xnoprt
EW_MAXIMIZE	PA3	Xnosym
EW_MINIMIZE	PRINT	Xnum
EW_RESTORE	RESET	Xprogck
E_BADSCRID	RIGHT_ARROW	Xprtbsy
E_CANTINIT	SS_APPLICATION	Xprtvrbs
E_COMMAND	SS_CONTROL	Xready
E_FLDNUM	SS_DISCONNECTED	Xsystem
E_INUSE	SS_UNOWNED	Xunauth
E_LINENUM	ST_3270_HOST	Xunknown
E_NOFLDS	ST_3270_PRINTER	Xwhat
E_NOTAVAIL	ST_5250_HOST	Xwhtnum
E_NOTCONN		

20.8 Sample Program Using 3270 Screens

Sample 3270 Screen "A":

MIN01001	ABC MANUFACTURING COMPANY
REPORT SELECTION	
<u>Report Name</u>	<u>ID Number</u>
Production Statistics	PS4471
Defect Reporting	DF1678
Warehouse Inventory	WI3672
Manufacturing Lead Time	MF2599
Enter Report ID Number: _____	

Sample 3270 Screen "B":

DF167801

Divn. : Acme ABC MANUFACTURING COMPANY
Report No: DF1678 DEFECTIVE PARTS REPORTING SYSTEM

Part Description	JAN	FEB	MAR	APR	MAY
3061 Cam Shaft	12	11	9	10	14
Bearing Ring	11	9	11	8	7
Time Gear - Large	11	19	14	16	15
Time Gear - Small	12	15	16	18	14
Cam Shaft Dowel Pin	15	16	13	16	15
Cam Shaft Lock Nut	9	11	12	12	13
Valve Bushing	12	9	7	9	19
TOTAL	82	90	82	89	97

- More -

PF1 = NEXT PAGE PF7 = RETURN TO MENU

20.8.1 Sample Program Using EASEL OS/2 EE 3270 Support

```
screen size device units      #use resolution of device
font "system"
module M3270

include "accel.inc"          #include accelerator keys for action bar
include "message.inc"        #include message box routines

boolean variable More is false

string variable LineChosen
                PartName
                ReplyIs
                MessageIs

# ACTION BAR DEFINITION

action bar ActionBar is
  pulldown File text "~File"
    disabled choice Open text "~Open..."
    disabled choice New text "~New"
    separator
    disabled choice Save text "~Save"
    disabled choice SaveAs text "Save ~As..."
    separator
    choice Exit text "E~xit\tF3"
      accelerator is KEY_F3
    end pulldown
  pulldown ReportList text "~Reports"
    choice Prodstat text "~Production Stats"
      parameter is "PS4471" #This is the ID No. (shown in
                           #Sample Screen A)
    choice Defect text "~Defect Reports"
      parameter is "DF1678"
    choice Warehouse text "~Warehouse Inventory"
      parameter is "WI3672"
    choice Manufacturing text "~Manufacturing"
      parameter is "MF2599"
    end pulldown
end action bar

# OBJECT DEFINITIONS

white primary region Primary size 500 350 at position 75 75
  title bar "ABC Manufacturing Company"
  system menu
  action bar ActionBar
  minimize button
```

```

maximize button

# Establish a class of items that includes all of the Report
# choices:

item class RptSelect is Prodstat Defect Warehouse Manufacturing

# Create a textual region to show the selected report:

invisible blue textual region Text window size 60 columns 17 lines
  at position 120 5 in desktop
  title bar "Requested Report"
  system menu
  vertical scroll bar
  border
  font "asc814"

# ACTION SUBROUTINES

# Copy current screen into Text textual region, and check if
# there are more screens to come:

action ReadReportPage is
  copy "Text" to TextRegionName
  action ReadScreen          # Appends screen data to Text
  copy 22 to Row
  action ReadLine
  switch LineText is
    case char "- More -" is
      copy true to More
    default is
      copy false to More
  end switch

# At first page of report, read all pages into Text textual
# region:

action GetPages is
  action ReadReportPage
  while (More) loop
    copy 1 to PFkeyNumber    # Get next page if "- More -"
    action PressPFkey        # is displayed
    action WatchAndWait
    if (WatchGaveUp) then
      copy false to More
    else
      action ReadReportPage

```

```

        end if
    end loop
    copy 7 to PFkeyNumber          # Return to menu screen
    action PressPFkey
    action WatchAndWait

# Set up the watch criteria for determining when a screen
# has been received:

action SetupWatch
    action DefineWatch
    copy 125 to SettleTime        # Set 1.25 seconds settle time
    copy 1000 to GiveUpTime      # Set 10 seconds maximum for host
                                # to respond
    action WatchForNoX           # Criteria if no X on Operator
                                # Information Area

# Log off from the host
action Logoff is
    # The code to log off
    # from the host should
    # be placed here

# RESPONSE DEFINITIONS

response to start
    copy "A" to SessionName
    action Init3270
    action SetupWatch
# Code to log on to the mainframe
# should be placed here

response to item RptSelect        #class of reports
    copy parameter to Keystrokes
    action TypeString
    action PressENTER
    action WatchAndWait
    if (WatchGaveUp) then        #put up message box with problem
        copy "Could not get report " Keystrokes "." to MessageIs
        copy (ReplyToMessage("System Problems," MessageIs,
            MessageOK, MessageOK, MessageIconHand)) to ReplyIs
        make Text invisible
    else
        action GetPages
        make Text visible
    end if

response to item Exit
    action Logoff
    action Disconnect

```

```
action Stop3270  
exit
```

```
response to Primary on close  
    action Logoff  
    action Disconnect  
    action Stop3270  
    exit
```

Chapter 21. 5250 Support

21.1 Introduction

21.1.1 EASEL OS/2 EE in the 5250 Environment

5250 is the generic name of a family of IBM terminals, controllers, and associated software. 5250 applications transmit information on a screen-by-screen basis.

With 5250 Support for EASEL OS/2 EE, a full screen of information is held in a screen buffer, where its data can be accessed. 5250 Support for EASEL OS/2 EE provides facilities that allow you to examine data in the buffer, place data into specified fields, transfer data to the host, and execute other functions that can be performed by a 5250 terminal. These functions are accomplished by calling the **M5250** module action routines described in this chapter.

21.1.2 Components of 5250 Support for EASEL OS/2 EE

The following 5250 Support components allow you to create an EASEL OS/2 EE interface to a 5250 application.

21.1.2.1 M5250 Module

The **M5250** module defines EASEL OS/2 EE action routines specific to the 5250 environment, and EASEL OS/2 EE variables used by the **M5250** action routines. These action routines:

- Read the contents of the screen buffer into an EASEL OS/2 EE textual region.
- Read a line into a string variable.
- Read a field's contents into a string variable.
- Write the contents of a string variable to a field.
- Send data to the controller.
- Simulate the pressing of a 5250 terminal's special function keys.

21.1.2.2 A5250.EXE Local Application

5250 Support for EASEL OS/2 EE uses the A5250.EXE local application internally, to communicate with host applications via Communications Manager (CM) and EHLLAPI.

21.1.2.3 FIELDS52.EXE Development Tool

The FIELDS52.EXE EASEL OS/2 EE development tool provides you with essential information about each field on the current screen in the 5250 screen buffer.

21.1.3 Using the M5250 Module: Required Procedures

The following steps are required to successfully implement an EASEL OS/2 EE program that uses the **M5250** module.

1. Install IBM OS/2 Extended Edition Communications Manager (CM), if it is not already installed. Configure CM for one or more 5250 terminal emulation sessions.
2. Make sure 5250 terminal emulation is started before running an EASEL OS/2 EE program that uses the **M5250** module, or before running the FIELDS52.EXE utility. 5250 terminal emulation can be started from the CM "Start Communications" menu (or CM can be configured to automatically start 5250 communications when it is started). Refer to your OS/2 Extended Edition manual, *Using Communications Manager*, for more information.
3. Where necessary, examine the output generated by FIELDS52.EXE to identify the precise contents of the buffer.
4. Write a test program that:
 - a. References the **M5250** module.
 - b. Initializes 5250 communications and connects to one or more sessions.
 - c. Synchronizes the timing of 5250 and EASEL OS/2 EE via **M5250** action statements, so that the screen buffer is filled before actions on it are taken.
 - d. Reads specific buffer contents into EASEL OS/2 EE string variables and textual regions, and communicates with the host via **M5250** action routines.
5. Write the final program, incorporating step 4 with other EASEL OS/2 EE action and response statements.

21.1.4 Referencing the M5250 Module

To reference the **M5250** module from an EASEL OS/2 EE program, include the following statement at the very beginning of the program:

```
module M5250
```

21.1.5 Sample Program

The following example shows a complete EASEL OS/2 EE program for a 5250 application that contains one full screen buffer.

```
module M5250                                # References the 5250 module.

primary textual region Report                # Defines a textual region in
  window size 80 columns 24 lines           # which to display the data.
  at position 0 100
  title bar "Report"
  system menu
  white border
  font "asc814"

response to start
  copy "A" to M52_SessionName               # Initial session to connect.
  action M52_Init                           # Initializes 5250 Commun.
  copy "Report" to M52_TextRegionName        # Copies the name of the
                                              # textual region that
                                              # will store the screen
                                              # data to M52_TextRegionName
                                              # (a predefined variable).
  action M52_ReadScreen                     # Copies the contents of
                                              # the 5250 screen buffer
                                              # into the textual region.

response to Report on close
  action M52_Disconnect                     # Disconnects from session A.
  action M52_Stop                           # Stops 5250 Communications.
  exit                                      # Exits EASEL OS/2 EE.
```

21.2 5250 Session and Screen Concepts

21.2.1 Session Names

CM allows you to create up to five logical 5250 terminal sessions, i.e., your workstation can emulate up to five 5250 display terminals at one time.

M5250 action routines allow EASEL OS/2 EE to be in control of one or more of these sessions. Each session configured by CM has two names:

- A long session name
- A short session name.

The long session name is up to eight characters long. The short session name is a single letter, such as "A" or "D." Only short session names are used by **M5250** action routines for denoting sessions. 3270 host sessions cannot be accessed by **M5250** action routines, but both 3270 and 5250 host sessions can be accessed in the same program by using the **M3270** and **M5250** action routines, respectively.

21.2.2 Session States

From the point of view of EASEL OS/2 EE, each configured session, once it has been started by CM, can be in one of three states:

- Disconnected
- Connected, not active
- Connected and active.

21.2.2.1 Disconnected

When disconnected, the session is not controlled by EASEL OS/2 EE. It may, however, currently be connected by another process.

21.2.2.2 Connected, Not Active

When connected but not active, the session is controlled by EASEL OS/2 EE. However, it is not the session acted upon by the set of **M5250** action routines that read, write, or send data. Other running processes are prevented from connecting to this session.

21.2.2.3 Connected and Active

When connected and active, the session is controlled by EASEL OS/2 EE and is acted upon by the set of **M5250** action routines that read, write or send data. Other running processes are prevented from connecting to this session.

At any given time, EASEL OS/2 EE may be connected to none, some, or all of the started 5250 host sessions. If it is connected to one or more sessions, only one of these may be the active connected session; the rest, if any, are not active. If there is no active connected session, an error condition will result from calling any **M5250** action routine that reads, writes, or sends data, returns status information, etc.

The **M5250** action routines that connect to or disconnect from 5250 host sessions are:

- M52_Init**
- M52_Stop**
- M52_Connect**
- M52_Disconnect**

21.2.3 Field-oriented and Line-oriented Screens

The 5250 screen buffer contains 24 lines by 80 columns and is generally field-oriented (also called formatted), or organized by fields.

Most of the **M5250** action routines are written to access field-oriented screens. 5250 Support for EASEL OS/2 EE accesses the screen buffer by first assigning sequential numbers to each field in the buffer. By specifying a field number, you can retrieve that field's contents and associated attributes, or write text into the specified field.

A 5250 application sometimes contains screens wherein each line is a field of its own, or contains screens that have no fields at all. These are called line-oriented (or unformatted) screens. There are two **M5250** action routines used to access line-oriented screens:

- M52_ReadLine**
- M52_FindLastNonBlankLine**

You can use these action routines to access line-oriented or field-oriented screens, but you cannot use the field-oriented action routines to access line-oriented screens.

21.2.3.1 Field-oriented (Formatted) Screens

By definition, a field is at least one character in length. A field can wrap around to the next line, or even to the beginning of the screen buffer. Each field is preceded by a field attribute character, which defines the location and attributes of the field, but is not part of the field itself. Each field extends up to the next field attribute character, which defines the next field in the screen. The following attributes are defined by the field attribute character:

Attribute	Description
protected/unprotected	Indicates whether the field is protected. A protected field cannot be modified (written) by the user.
modified/unmodified	Indicates whether the field was modified (written).
display/nondisplay	Indicates whether the field is displayed on a 5250 terminal.
Intensified/normal	Indicates the brightness of the characters in the field.

The field attribute character also indicates one of the following field types:

Field Type	Description
Alphanumeric	All characters are allowed.
Alphabetic only	Uppercase and lowercase letters, comma, period, hyphen, blank, and Dup key are allowed.
Numeric shift	Automatic shift for numerics is enabled.
Numeric only	Digits 0-9, comma, period, plus, minus, blank, and Dup key are allowed.
Digits only	Digits 0-9 and Dup key are allowed.
Magnetic stripe reader	Data can be entered only via a magnetic stripe reader.
Signed numeric	Digits 0-9, plus, minus, and Dup key are allowed.

The 5250 application also assigns colors to fields, along with any or none of these visual attributes:

- Reverse video
- Underscore
- Blink
- Column separators.

Of these, EASEL OS/2 EE supports the color and reverse video attributes, so that you can examine characters for foreground and background colors, once they have been read into a colored textual region.

Character positions within the 5250 buffer are addressed by row and column. Row numbers go from 1 to 24, from top to bottom; column numbers go from 1 to 80, from left to right. The following examples illustrate how a character position is expressed as an ordered pair of numbers:

[1,2] = row 1, column 2

[24,80] = row 24, column 80

Sometimes a host message is temporarily displayed on the 25th row (which is normally where the Operator Information Area is displayed). In this situation, row numbers go from 1 to 25. (See 21.4.5.8, "M52_ReadLine" on page 21-28.)

Characters within the 5250 Operator Information Area (the bottom line of the host system when there is no message being displayed) cannot be accessed directly via 5250 Support for EASEL OS/2 EE. (See 21.4.10.2, "M52_CheckIndicators" on page 21-56.)

21.3 Retrieving Field Information with FIELDS52.EXE

FIELDS52.EXE is a development tool that generates a report containing up to three sections:

- Information about the current screen
- Field summary
- Full field contents.

21.3.1 Reading the FIELDS52.EXE Report

21.3.1.1 Information About the Current Screen

The information about the current screen includes the following information:

- Index by column number
- Entire screen contents
- Number of fields in the screen
- Current cursor position, by row and column.

21.3.1.2 Field Summary

The field summary includes the following information for each field in the screen:

- Field number
- Starting and ending field positions, by row and column
- Field length
- Field attributes
- First 26 characters in the field.

21.3.1.3 Full Field Contents

The full field contents lists all characters in each field.

By default, only the first two sections are reported. In general, the field summary section should provide all the field information you need. (See 21.3.3, "FIELDS52.EXE Sample Output" on page 21-11.)

21.3.2 Using FIELDS52.EXE

To use FIELDS52.EXE:

1. Access the 5250 host application (using the CM 5250 emulator), and go to the screen for which you need the field information.
2. Activate an OS/2 fullscreen or windowed command prompt.

3. At the OS/2 command prompt, invoke FIELDS52.EXE as follows:

Model

fields52 [OPTIONS] [OUTPUT_FILENAME]

Where: **OPTIONS** is a string containing any of the following parameters:

Parameter Meaning

- sNAME** Report is for the 5250 terminal session represented by the specified single-letter short session name. This parameter defaults to the first 5250 host session listed when FIELDS52.EXE is run with the "-L" option. You should always specify this parameter when CM is configured for more than one 5250 terminal emulation session.
- b** Report contains screen information section only.
- f** Report contains all three sections.
- L** Report lists all 5250 and 3270 host sessions started by CM. No field information is given. For each session, its short name, long name, screen dimensions, and type are shown. Sessions that have not been started are not listed.

OUTPUT_FILENAME

The name and extension of the OS/2 file in which the FIELDS52 report is placed. If OUTPUT_FILENAME is omitted, the report is listed on your workstation's display, and the screen information section does not show the screen contents.

Note: If "-L" is specified, this parameter is ignored; the sessions are always listed on your workstation's display.

Examples: fields52

fields52 login.fld

fields52 -b news4.fld

fields52 -f -sB product.fld

21.3.3 FIELDS52.EXE Sample Output

The following is sample output for the -L parameter:

FIELDS52 utility - Version E1.1 (EHLAPI)
International Business Machines Corporation
(C) Copyright Interactive Images, Inc. 1987, 1990
All Rights Reserved.

Number of started host sessions = 3

Short Name	Long Name	Rows	Columns	Type
-----	-----	----	-----	-----
A	HOST1	24	80	5250 Host
B	HOST2	24	80	5250 Host
C	HOST3	43	80	5250 Host

The following two pages show sample output from **FIELDS52.EXE** when no options are specified. The area shown within the double dashed lines is how the entire screen would appear on a terminal. The first double dashed line is preceded by a column number index.

FIELDS52 utility - Version E1.1 (EHLAPI)
International Business Machines Corporation
(C) Copyright Interactive Images, Inc. 1987, 1990
All Rights Reserved.

Session short name = F

0000000001111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

=====

Menu: *MASTER	Master Menu	12/09/89
		16:32:49

Option:

1. 3270 EMULATION
2. Password Change
3. Start Woburn Printer 1
4. Display Woburn Printer
5. Display Woburn Reports
6. Cancel Woburn Writer
7. Woburn Line Status
8. Display QINTER Subsystem
9. Send Break Message

Cmd1 = Signoff	Cmd7= Submitted Jobs	Cmd6 = Messages *WRKSTN
Cmd13= Signoff	Cmd8= Display Job	Cmd18= Messages WOBNWS1

=====

Screen contains 33 fields.
Cursor is at [3,11]

Field summary:

Unp = Unprotected	AN = Alphanumeric	DO = Digits only
Nds = Nondisplay	AB = Alphabetic only	MS = Mag stripe data only
Int = Intensified	NS = Numeric Shift	SN = Signed numeric
Mod = Modified	NO = Numeric Only	

FLD#	Start	End	Length	Unp	NDS	Int	Mod	Type	First 26 Characters
1	[1,2]	[1,6]	5			X		AN	[Menu]
2	[1,8]	[1,17]	10			X		AN	[*MASTER]
3	[1,19]	[1,27]	9					AN	[]
4	[1,29]	[1,53]	25			X		AN	[Master Menu]
5	[1,55]	[1,71]	17					AN	[]
6	[1,73]	[1,80]	8					AN	[12/09/89]
7	[2,2]	[2,6]	5					AN	[]
8	[2,8]	[2,17]	10					AN	[]
9	[2,19]	[2,71]	53					AN	[]
10	[2,73]	[2,80]	8					AN	[16:32:49]
11	[3,3]	[3,9]	7			X		AN	[Option:]
12	[3,11]	[3,12]	2	X				DO	[]
13	[3,14]	[3,27]	14					AN	[]
14	[3,29]	[3,53]	25					AN	[]
15	[3,55]	[4,2]	28					AN	[]
16	[4,4]	[4,78]	75					AN	[]
17	[5,1]	[19,80]	1200			X		AN	[1. 3270 EMULATION
18	[20,2]	[20,80]	79					AN	[]
19	[21,2]	[21,7]	6					AN	[Cmd1 =2]
20	[21,9]	[21,18]	10					AN	[Signoff]
21	[21,20]	[21,25]	6					AN	[]
22	[21,27]	[21,46]	20					AN	[Cmd7= Submitted Jobs]
23	[21,48]	[21,52]	5					AN	[]
24	[21,54]	[21,76]	23					AN	[Cmd6 = Messages *WRKSTN]
25	[21,78]	[21,80]	3					AN	[]
26	[22,2]	[22,7]	6					AN	[Cmd13=]
27	[22,9]	[22,18]	10					AN	[Signoff]
28	[22,20]	[22,25]	6					AN	[]
29	[22,27]	[22,43]	17					AN	[Cmd8= Display Job]
30	[22,45]	[22,52]	8					AN	[]
31	[22,54]	[22,68]	15					AN	[CMD18= Messages]
32	[22,70]	[22,79]	10					AN	[WOBNWS1]
33	[24,2]	[24,80]	159					AN	[]

21.4 M5250 Module Action Routines

The **M5250** module defines a set of variables and action routines. Each action routine is associated with two sets of variables: input variables and output variables. By copying values to the input variables of a specified action routine and then calling the action routine, data is returned in the output variables of the action routine.

In the following example, the program reads a street address contained in the third field of the 5250 screen buffer, and displays it as part of a graphical object on the EASEL OS/2 EE screen.

```
copy 3 to M52_FieldNumber # input variable for
                           # M52_ReadField"
action M52_ReadField      # call action routine
change AddressKey to      # AddressKey is defined
                           # in program
    text M52_FieldText    # output variable from
                           # M52_ReadField
:
```

21.4.1 M5250 Action Routines and EASEL OS/2 EE Built-in Functions

All **M5250** action routines except the following reset all EASEL OS/2 EE response inquiry built-in functions:

M52_DefineWatch	M52_WatchForIIOff
M52_WatchForCursorAt	M52_WatchForChar
M52_WatchForCursorNotAt	M52_WatchForNotChar

Example:

```
response to Key_X          # This will not work,
    action M52_PressENTER  # because "object" was
    make object green      # reset by "M52_PressENTER".
```

If you need to preserve the value of an EASEL OS/2 EE response inquiry built-in function, copy the value to a temporary variable before calling any **M5250** action routines. Then the temporary variable name can be used whenever the built-in function value is required. The following example preserves the value of the **object** built-in function in the string variable **Temp**.

```
copy object to Temp
:
action ...
:
make Temp invisible
```

21.4.2 Summary of M5250 Action Routines

The following table summarizes the action routines provided in the **M5250** module. Following the summary, each action is described in detail. Each description lists the action routine's associated input variables and output variables, notes error conditions, gives sample code where applicable, and refers you to associated action routines.

Action Routine	Usage
<i>Starting/Stopping:</i>	
M52_Init	Initializes M5250 variables and starts A5250 local application.
M52_Stop	Stops A5250 local application.
<i>Connecting and Disconnecting Sessions:</i>	
M52_Connect	Connects to a specified session.
M52_Disconnect	Disconnects from the active connected session.
<i>Examining the Buffer:</i>	
M52_ScanScreen	Field-oriented. Records locations and sizes of all fields in the 5250 screen buffer; returns a variable containing the number of fields.
M52_ReadField	Field-oriented. Copies the contents of the specified field into a string variable.
M52_GetFieldLength	Field-oriented. Returns the length of a specified field.
M52_GetFieldAttributes	Sets output variables (M52_Protected , M52_Modified , M52_NonDisplay , M52_Intensified , M52_FieldType) based on field attributes of a specified field.
M52_GetCursorPosition	Returns the row and column of the current cursor position.
M52_GetFieldNumber	Field-oriented. Locates the 5250 field number for a specified screen position.
M52_GetFieldPosition	Field-oriented. Returns the starting row and column position of a specified field.
M52_ReadLine	Line-oriented. Copies the contents of a line into a string variable.
M52_FindLastNonBlankLine	Line-oriented. Returns the last non-blank line at or above a specified starting line.
M52_ReadScreen	Copies the contents of the 5250 screen buffer into a textual region.

M52_ReadColoredScreen	Copies the contents of the 5250 screen buffer, with colors, into a colored textual region.
<i>Synchronization:</i>	
M52_InformWhenIIOff	Activates a response you have defined to signal when a new screen is ready. Allows EASEL OS/2 EE to take other responses while waiting for the new screen to be ready.
M52_WaitForIIOff	Waits until the new screen is ready before allowing EASEL OS/2 EE to take other responses.
M52_DefineWatch	Called in preparation for one or more of the “watch for” action routines to be called.
M52_WatchForCursorAt	Queues a command for watching the 5250 screen until the cursor is located at a specified row and column position.
M52_WatchForCursorNotAt	Queues a command for watching the 5250 screen until the cursor is located anywhere except a specified row and column position.
M52_WatchForIIOff	Queues a command for watching the Operator Information Area until the Input Inhibited indicator remains off for a specified period of time.
M52_WatchForChar	Queues a command for watching the 5250 screen until a specified character appears at a specified row and column position.
M52_WatchForNotChar	Queues a command for watching the 5250 screen until any character other than a specified character appears at a specified row and column position.
M52_Watch	Executes previously queued “watch” command(s) and activates a response you have defined to signal when a new screen is ready. Allows EASEL OS/2 EE to take other responses while “watch” is in effect.
M52_WatchAndWait	Executes previously queued “watch” command(s) and waits until a new screen is ready before allowing EASEL OS/2 EE to take other responses.

Communicating with the Host:

M52_PressKey	Emulates pressing a specified “special” key sequence on a 5250 terminal.
M52_PressENTER	Emulates pressing the ENTER key on a 5250 terminal.

M52_PressCMDkey	Emulates pressing a CMD key sequence on a 5250 terminal.
M52_TypeString	Emulates pressing keys corresponding to any of the printing ASCII characters.
M52_EnterString	Same as M52_TypeString followed by M52_PressENTER .

Miscellaneous Action Routines:

M52_WriteField	Copies a string into the specified field.
M52_CheckIndicators	Returns status of indicators (System Available, Input Inhibited, Message Waiting) in the boolean variables M52_SystemAvailable , M52_InputInhibited , and M52_MessageWaiting .
M52_GetSessionInfo	Returns session type and screen dimensions of a specified session.
M52_PrintScreen	Prints the current 5250 screen buffer contents. Used for debugging.

21.4.3 Starting and Stopping 5250 Support for EASEL OS/2 EE

The following two commands start and stop **A5250**, the local application that allows EASEL OS/2 EE to access the 5250 emulation adapter.

21.4.3.1 M52_Init

<i>Syntax:</i>	action M52_Init	
<i>Input</i>	M52_SessionName	(string)
<i>Variables:</i>		
<i>Output</i>	None	
<i>Variables:</i>		
<i>Remarks:</i>	M52_Init copies initial values to global variables used by the other M5250 action routines, starts the local application A5250, and connects to the session specified in M52_SessionName. M52_SessionName must specify a 5250 host session short name. Use action M52_Connect if you need to connect to additional sessions.	
<i>Example:</i>	<pre>response to start # Initialize 5250 communications and connect to # session B. copy "B" to M52_SessionName action M52_Init</pre>	
<i>Errors:</i>	M52_E_CANTINIT M52_E_INUSE M52_E_SESSNAME	Cannot initialize Session is already in use Invalid session name
<i>See Also:</i>	M52_Connect	

21.4.3.2 M52_Stop

Syntax: **action M52_Stop**

Input None

Variables:

Output None

Variables:

Remarks: **M52_Stop** should be called when your program has completed 5250 communications. You should ensure that all connected sessions get logged off. Although **M52_Stop** will disconnect all connected sessions, it is good programming practice to call action **M52_Disconnect** for each connected session first.

Example: response to StopSign
 copy "LOGOFF" to M52_Keystrokes
 action M52_EnterString
 action M52_Disconnect
 action M52_Stop
 exit

See Also: **M52_Disconnect**

21.4.4 Connecting and Disconnecting Sessions

21.4.4.1 M52_Connect

<i>Syntax:</i>	action M52_Connect	
<i>Input Variables:</i>	M52_SessionName	(string)
<i>Output Variables:</i>	None	
<i>Remarks:</i>	EASEL OS/2 EE connects to the session specified by M52_SessionName. This session becomes the active connected session. The previous active connected session, if any, remains connected, but it is no longer active. It is not an error to specify a session that is already connected; in fact, this is how you reactivate it. M52_SessionName must specify a 5250 host session short name.	
<i>Example:</i>	<pre>copy "A" to M52_SessionName action M52_Connect # Session A is now the active connected session. copy "B" to M52_SessionName action M52_Connect # Session A connected; session B is the active # connected session. copy "A" to M52_SessionName action M52_Connect # Session B connected; session A is the active # connected session.</pre>	
<i>Errors:</i>	M52_E_INUSE M52_E_SESSNAME	Session is already in use Invalid session name
<i>See Also:</i>	M52_Init	

21.4.4.2 M52_Disconnect

Syntax: **action M52_Disconnect**

Input None

Variables:

Output None

Variables:

Remarks: **M52_Disconnect** disconnects from the active connected session. To disconnect from a session that is connected but not active, you must first make it active by calling **action M52_Connect**. When **action M52_Disconnect** returns, there is no active connected session (although there may still be connected sessions).

Example: # Disconnect from sessions A and B. The current
 # active connected session is A.
 action M52_Disconnect # Disconnect from A.
 copy "B" to M52_SessionName
 action M52_Connect # Make B active.
 action M52_Disconnect # Disconnect from B.

Errors: M52_E_NOTCONN Not connected to an active session.

See Also: **M52_Stop**

21.4.5 Examining the Buffer

Before writing a line of 5250 Support code, you should become familiar with the behavior of the host 5250 application. You should know the contents of and proper responses for each 5250 screen.

The following action routines allow you to interrogate the buffer and copy its contents to variables, and use them just as you would use any other EASEL OS/2 EE variables.

21.4.5.1 M52_ScanScreen

Syntax: **action M52_ScanScreen**

Input None

Variables:

Output **M52_FieldCount** (integer)

Variables:

Remarks: **M52_ScanScreen** scans the screen buffer and records the locations and sizes of all the fields. Upon completion of this routine, the output variable **M52_FieldCount** contains the total number of fields in the screen buffer. You cannot directly access the field information that is recorded, but the information is available through and used by many of the action routines described in this section.

You must call **M52_ScanScreen** after the contents of the screen have changed, if the new screen's format is different, and if you are referencing this screen by fields.

Once **M52_ScanScreen** is called, any field can be specified by setting the variable **M52_FieldNumber** to a value between **1** and **M52_FieldCount**, inclusive.

Example: response to start
 copy "B" to M52_SessionName
 action M52_Init
 action M52_ScanScreen
 :

21.4.5.2 M52_ReadField

<i>Syntax:</i>	action M52_ReadField	
<i>Input</i>	M52_FieldNumber	(integer)
<i>Variables:</i>		
<i>Output</i>	M52_FieldText	(string)
<i>Variables:</i>		
<i>Remarks:</i>	M52_ReadField copies the contents of field number M52_FieldNumber into string M52_FieldText . You can use M52_ReadField to examine the contents of nondisplay fields.	
<i>Example:</i>	# In this example, Date_Window is a textual region # defined in the program. : copy 4 to M52_FieldNumber action M52_ReadField add to Date_Window insert M52_FieldText	
<i>Errors:</i>	M52_E_FLDNUM	Invalid field number
	M52_E_NOFLDS	Screen contains no fields or has not been scanned

21.4.5.3 M52_GetFieldLength

Syntax: **action M52_GetFieldLength**

Input **M52_FieldNumber** (integer)

Variables:

Output **M52_FieldLength** (integer)

Variables:

Remarks: **M52_GetFieldLength** returns the length (**M52_FieldLength**) of a specified field (**M52_FieldNumber**). It returns zero if there is an error.

Example: copy 9 to M52_FieldNumber
 action M52_GetFieldLength
 if (M52_FieldLength = 8) then
 :
 end if

<i>Errors:</i>	M52_E_FLDNUM	Invalid field number
	M52_E_NOFLDS	Screen contains no fields or has not been scanned

21.4.5.4 M52_GetFieldAttributes

Syntax: **action M52_GetFieldAttributes**

Input **M52_FieldNumber** (integer)

Variables:

Output **M52_Protected** (boolean)

Variables: **M52_Modified** (boolean)

M52_NonDisplay (boolean)

M52_Intensified (boolean)

M52_FieldType (integer)

Remarks: **M52_GetFieldAttributes** sets the above output boolean variables to **true** or **false**, based upon the attribute character associated with the specified field. In addition, it sets the above output integer variable, **M52_FieldType**, to a value that denotes the type of data that may be contained in the field. (See Field-Oriented Screens for more information about field types.) Use the following constants for testing the value of **M52_FieldType**:

M52_FT_ALPHANUMERIC
M52_FT_ALPHA_ONLY
M52_FT_NUMERIC_SHIFT
M52_FT_NUMERIC_ONLY
M52_FT_DIGITS_ONLY
M52_FT_MAG_STRIPE
M52_FT_SIGNED_NUMERIC

If there is an error, all the booleans return **false** and **M52_FieldType** is set to **M52_FT_ALPHANUMERIC**.

Example: action M52_ScanScreen
 copy 6 to M52_FieldNumber
 action M52_GetFieldAttributes
 if (not M52_Protected and
 M52_FieldType = M52_FT_DIGITS_ONLY) then
 :
 end if

<i>Errors:</i>	M52_E_FLDNUM	Invalid field number
	M52_E_NOFLDS	Screen contains no fields or has not been scanned

21.4.5.5 M52_GetCursorPosition

Syntax: **action M52_GetCursorPosition**

Input None

Variables:

<i>Output</i>	M52_Row	(integer)
<i>Variables:</i>	M52_Column	(integer)

Remarks: **M52_GetCursorPosition** returns the current 5250 screen cursor position by row (**M52_Row**) and column (**M52_Column**).

Example: action M52_GetCursorPosition
 if ((M52_Row = 24) and (M52_Column = 56)) then
 make Intro_Screen visible
 :
 :

See Also: **M52_GetFieldNumber**

21.4.5.6 M52_GetFieldNumber

Syntax: **action M52_GetFieldNumber**

Input **M52_Row** (integer)

Variables: **M52_Column** (integer)

Output **M52_FieldNumber** (integer)

Variables:

Remarks: **M52_GetFieldNumber** finds the field number at the specified screen position. Screen positions start at (1,1) in the upper left corner of the screen.

If the given row (**M52_Row**) and column (**M52_Column**) are in the middle of a field, its field number is returned. If the given row and column are not inside a field, the number of the closest field further down the screen is returned. If there is an error, **M52_FieldNumber** contains zero.

Example: action M52_ScanScreen
 action M52_GetCursorPosition # Returns row and column.
 action M52_GetFieldNumber # Uses row and column.
 if (M52_FieldNumber = 7) then
 :
 end if

Errors: **M52_E_NOFLDS** Screen contains no fields or has
 not been scanned
 M52_E_ROWCOL (Row, Column) is not on screen

See Also: **M52_GetCursorPosition**

21.4.5.7 M52_GetFieldPosition

<i>Syntax:</i>	action M52_GetFieldPosition	
<i>Input</i>	M52_FieldNumber	(integer)
<i>Variables:</i>		
<i>Output</i>	M52_Row	(integer)
<i>Variables:</i>	M52_Column	(integer)
<i>Remarks:</i>	M52_GetFieldPosition returns the starting row (M52_Row) and column (M52_Column) position of a specified field (M52_FieldNumber). Position (1,1) is the upper left corner of the screen buffer. If there is an error, (0,0) is returned.	
<i>Example:</i>	copy 7 to M52_FieldNumber action M52_GetFieldPosition if ((M52_Row = 2) and (M52_Column = 28)) then : end if	
<i>Errors:</i>	M52_E_FLDNUM	Invalid field number
	M52_E_NOFLDS	Screen contains no fields or has not been scanned

21.4.5.8 M52_ReadLine

Syntax: **action M52_ReadLine**

Input **M52_Row** (integer)
Variables:

Output **M52_LineText** (string)
Variables:

Remarks: **M52_ReadLine** copies the contents of line number
 M52_Row of the screen buffer into the string variable
 M52_LineText.

M52_ReadLine is provided for use on screens that are line-oriented (unformatted). If it is applied to a screen that contains fields, the field information is ignored.

Reading the 25th Row: Under CM, the 5250 Work Station Feature supports a presentation space (screen buffer) size of 24 rows by 80 columns. In some instances, a 25th row is displayed. This occurs when either an error message from the host is displayed or when the operator presses the SysReq key.

You can read the contents of the 25th row by calling **action M52_ReadLine** after copying the value 25 to the input variable **M52_Row**. If a 25th row is currently being displayed, its contents will be copied to the string variable **M52_LineText**. If no 25th row is being displayed, **M52_ReadLine** will return with **M52_LineText** containing a null string ("").

M52_ReadLine is the only **M5250** action that allows you to specify row 25.

Example: response to start
 copy "D" to M52_SessionName
 action M52_Init
 copy 1 to M52_Row
 action M52_ReadLine
 extract from M52_LineText
 take word M52_ScreenID
 if (M52_ScreenID="MIN01001") then
 make Intro_Region visible
 end if

Errors: **M52_E_LINENUM** Invalid line number

See Also: **M52_FindLastNonBlankLine**

21.4.5.9 M52_FindLastNonBlankLine

Syntax: **action M52_FindLastNonBlankLine**

Input **M52_StartLine** (integer)

Variables:

Output **M52_Row** (integer)

Variables:

Remarks: **M52_FindLastNonBlankLine** returns the number of the last non-blank line (**M52_Row**) on the screen at or above **M52_StartLine**. It returns the value zero if all lines at or above **M52_StartLine** are blank or if there is an error. The initial value of **M52_StartLine** is 24.

M52_FindLastNonBlankLine is provided for use on screens that are line-oriented (unformatted), providing you with an easy way to locate the most recent prompt or message displayed by the application. If it is applied to a screen that contains fields, the field information is ignored.

Example: action CheckOptions is
 copy 22 to M52_StartLine # Search starts at line 22.
 action M52_FindLastNonBlankLine
 action M52_ReadLine
 switch M52_LineText is
 case char "-More-" is
 make More_Key visible # More_Key sends
 # commands to host
 # to show next screen
 case char "-End of Data-" is
 make Another_Key visible # Another_Key sends
 # commands to host
 # to show menu
 default is
 make Another_Key visible # Generic response
 # to take care of
 # unexpected lines
 end switch

Errors: **M52_E_LINENUM** Invalid line number

See Also: **M52_ReadLine**

21.4.5.10 M52_ReadScreen

Syntax: **action M52_ReadScreen**

Input **M52_TopRow** (integer)
Variables: **M52_BottomRow** (integer)
 M52_LeftCol (integer)
 M52_RightCol (integer)
 M52_TextRegionName (string)

Output Text from the screen buffer is inserted into the textual
Variables: region **M52_TextRegionName**.

Remarks: **M52_ReadScreen** inserts the contents of the 5250 screen buffer into a textual region. You can specify any rectangular area within the buffer by setting values for **M52_TopRow**, **M52_BottomRow**, **M52_LeftCol**, and **M52_RightCol**. The initial values of these variables are 1 through 24 and 1 through 80.

You must clear the textual region before calling this action, unless you want text inserted into the textual region at the current cursor position.

Note: The contents of those fields with a non-display attribute on the 5250 terminal are visible in an EASEL OS/2 EE textual region.

Example: textual region Screen_1
 window size 80 columns 24 lines
 at position 100 100
 :
 clear Screen_1 # Clears textual region.
 copy 5 to M52_TopRow # Reads area of screen
 copy 20 to M52_BottomRow # within rows 5-20 and
 copy 10 to M52_LeftCol # columns 10-70.
 copy 70 to M52_RightCol
 copy "Screen_1" to M52_TextRegionName
 action M52_ReadScreen

Errors: **M52_E_SCRNAREA** Invalid screen area limits

See Also: **M52_ReadColoredScreen**

21.4.5.11 M52_ReadColoredScreen

Syntax: **action M52_ReadColoredScreen**

Input **M52_TopRow** (integer)

Variables: **M52_BottomRow** (integer)

M52_LeftCol (integer)

M52_RightCol (integer)

M52_TextRegionName (string)

Output Colored text from the screen buffer is inserted into the

Variables: colored textual region **M52_TextRegionName**.

Remarks: **M52_ReadColoredScreen** inserts the contents of the 5250 screen buffer into a colored textual region. You can specify any rectangular area within the buffer by setting values for **M52_TopRow**, **M52_BottomRow**, **M52_LeftCol**, and **M52_RightCol**. The initial values of these variables are 1 through 24 and 1 through 80.

You must clear the textual region before calling this action, unless you want text inserted into the textual region at the current cursor position.

The colors that appear in the textual region are the same as the colors displayed by the 5250 terminal emulation session, except in the following cases:

- Normal and intensified colors appear the same in the textual region. For example, the colors “cyan” and “light cyan” will be distinguishable in the terminal emulator, but will both appear as “cyan” in the colored textual region.
- The terminal emulator can simulate “underline” and “separator” field attributes by displaying an underline or some other symbol at character positions that actually contain nulls or blanks. The textual region never contains these simulated field attributes; it always contains the actual characters that reside in the screen buffer.

Note: The contents of those fields with a non-display attribute on the 5250 terminal are also not visible in an EASEL OS/2 EE colored textual region.

Example: colored textual region ColorCodedText
 window size 40 columns 22 lines
 at position 10 25
 :
 clear ColorCodedText # Clears textual region.
 copy 1 to M52_TopRow # Reads area of screen
 copy 22 to M52_BottomRow # within rows 1-22 and
 copy 41 to M52_LeftCol # columns 41-80.
 copy 80 to M52_RightCol
 copy "ColorCodedText" to M52_TextRegionName
 action M52_ReadColoredScreen

Errors: M52_E_SCRNAREA Invalid screen area limits

See Also: **M52_ReadScreen**

21.4.6 Synchronizing 5250 Support for EASEL OS/2 EE with the Host

Host transmissions to the screen buffer take time. If EASEL OS/2 EE examines the screen buffer too soon, it may attempt to perform actions based on an incomplete screen. The action routines described in this section allow your program to synchronize with the host application so that the screen buffer is examined only when it contains a complete screen.

There are two groups of action routines available for host synchronization:

- Based on Input Inhibited indicator
- **M5250** watch facility.

21.4.6.1 Input Inhibited Indicator

The first group of actions determine when the new screen is ready based on the Input Inhibited indicator, one of several indicators displayed within the Operator Information Area (the bottom line of a 5250 terminal or the CM 5250 emulator). The CM 5250 emulator displays the Input Inhibited indicator as a pair of uppercase Is (II), which appear in reverse video when the indicator is activated. The action routines in this group are **M52_InformWhenIIOff** and **M52_WaitForIIOff**. These actions are sufficient for synchronizing with most applications that reside on an AS/400 host.

21.4.6.2 M5250 Watch Facility

The second group of action routines are required for 3270 applications residing on a System/370-class mainframe and connected to the workstation through an AS/400 using a facility known as "3270 Device Emulation." Although these are 3270 applications, they are accessed via CM 5250 sessions (because the AS/400 converts the 3270 data stream to a 5250 data stream). Therefore, **M5250** action routines are required to access 3270 applications in this configuration. The second group of synchronization action routines is called the "**M5250 watch** facility."

21.4.7 Synchronizing with AS/400 Applications

21.4.7.1 M52_InformWhenIIOff

Syntax: **action M52_InformWhenIIOff**
Input **M52_GiveUpTime** (integer)
Variables:

Output None
Variables:

Remarks: **M52_InformWhenIIOff** watches the Input Inhibited indicator while allowing EASEL OS/2 EE to continue to respond to other tasks that do not pertain to the 5250 buffer (such as selections, keyboard input, and local application input), as long as those responses are within a guarded or resumable begin/end block.

M52_InformWhenIIOff completes either when the Input Inhibited indicator goes off, or after the period of time specified by **M52_GiveUpTime** has elapsed and the Input Inhibited indicator is still on. **M52_GiveUpTime** is in units of .01 seconds; the value 100 is equivalent to 1 second. The initial value is 3000 (30 seconds).

To detect the completion of **M52_InformWhenIIOff**, you must set up a guarded or resumable begin/end block with both of the following **response to line** statements, exactly as shown:

response to line "Wait done" from A5250
response to line "Wait gave up" from A5250

"Wait done" and "Wait gave up" are status messages from the A5250 local application. Normally, the message "Wait done" is received, unless **M52_GiveUpTime** elapses before the Input Inhibited indicator has gone off or an error has occurred.

We strongly recommend that you define the block as either **guarded** or **resumable**. If you want to restrict valid responses to be only those within the block, make the block **guarded**. If you want to have all other responses in outer blocks valid as well, make the block **resumable**. The begin/end block must have at least one **leave block** statement.

The **M52_InformWhenIIOff** action routine returns immediately after being called, but continues executing until one of the "wait" status messages is received from A5250. Calling any other **M5250** action routine while this action routine is in progress interrupts and cancels this action routine. No status message results from the canceled action routine.

Example:

```

:
action M52_PressENTER      # Pressing ENTER will cause the
                           # next screen to be displayed.
# Watch for the next screen to be completed:
copy 2000 to M52_GiveUpTime # Give up after 20 seconds.
action M52_InformWhenIIOff
begin guarded Block
    response to line "Wait done" from A5250      # When the
        copy "Mo_Report" to M52_TextRegionName  # screen is
        action M52_ReadScreen                   # complete,
                                                # use it.
    leave block                                # Include at least one
                                                # "leave block" statement
                                                # inside the block.

```

Example

```

# If wait did not complete within 20 seconds, have user
# try again.
response to line "Wait gave up" from A5250
    change Display_Message to
        text "System error, try again"
    make Display_Message visible
    leave block

    response to Menu_keys
    # Respond to selections while waiting for next
    # screen. All other active responses must be
    # inside this block.
:
end Block

```

See Also: **M52_WaitForIIOff**

21.4.7.2 M52_WaitForIIOff

Syntax: **action M52_WaitForIIOff**

Input **M52_GiveUpTime** (integer)
Variables:

Output **M52_WaitGaveUp** (boolean)
Variables:

Remarks: **M52_WaitForIIOff** waits for the Input Inhibited indicator to go off. It does not allow EASEL OS/2 EE to respond to other stimuli while **M52_WaitForIIOff** is running; therefore, no begin/end blocks need to be defined. Normally, it is called immediately after sending a keystroke that causes the host to transmit a new screen.

M52_WaitForIIOff is executed until either the Input Inhibited indicator goes off, or a period of time specified by **M52_GiveUpTime** has elapsed and the Input Inhibited indicator is still on.

M52_GiveUpTime is in units of .01 seconds; the value 100 is equivalent to 1 second. The initial value is 3000 (30 seconds).

M52_WaitForIIOff is executed until either the Input Inhibited indicator goes off, or a period of time specified by **M52_GiveUpTime** has elapsed and the Input Inhibited indicator is still on.

M52_WaitGaveUp is set to either **true** or **false**, depending on whether or not the **M52_GiveUpTime** has elapsed.

Example:

```
action M52_PressENTER      # Pressing ENTER will cause the
                           # next screen to be displayed
copy 2500 to M52_GiveUpTime # Wait for next screen
action M52_WaitForIIOff     # to be completed. If not
if (M52_WaitGaveUp) then    # completed in 25 seconds,
    change Display_Message to # have user try again.
    text "System error, try again"
    make Display_Message visible
else
    copy "Weekly_report" to # If completed
    M52_TextRegionName      # within 25
    action M52_ReadScreen   # seconds,
end if                      # use it.
```

See Also: **M52_InformWhenIIOff**

21.4.8 Synchronizing with Applications Via 3270 Device Emulation

For 3270 Device Emulation applications, the Input Inhibited indicator, by itself, is not always reliable for determining when the screen buffer contains a complete screen. For these applications, the **M5250** watch facility should be used. (A similar facility exists in 3270 Support for EASEL OS/2 EE.) This facility allows a program to detect and react to modifications made to the screen buffer during data transmission from the host application.

In order to use the watch facility, you first define a sequence of one or more watch commands. These define a set of conditions that must all be satisfied, in sequence, before the transmission of a screen is considered to be complete. You then invoke the appropriate action routine, which tells 5250 Support for EASEL OS/2 EE to start watching for those conditions to be met. For example, you can define a watch command sequence that expresses the following:

“Watch the cursor until it is not located at position row 24, column 6. Next, watch the Operator Information Area until the Input Inhibited indicator is off. Let me know when this is done.”

(This is the example shown in the description of the **M52_WatchAndWait** action routine.)

You can define watch commands to monitor any of the following:

1. The contents of a character position within the screen buffer
2. The cursor position
3. The Input Inhibited indicator (II) in the Operator Information Area

You can tailor the watch command sequences to the way your host operating system and application update the terminal screens. It will be easier for you to define appropriate **watch** command sequences by first answering the following questions on how the screens are updated:

1. When does the “II” in the Operator Information Area appear and disappear? Does it flicker as characters are written to the screen? Does it stay on continuously while waiting for the host to respond, or does it go on only when the host is transmitting?
2. Does the cursor position change from the previous screen? If so, when—after the “II” disappears, or just before?
3. Which characters will change from the previous screen? Which ones will stay the same? Will the screen be cleared before the next screen is displayed?

The answers to these questions will enable you to write EASEL OS/2 EE code that determines when the 5250 screen buffer has been completely updated.

You define a watch command sequence by first calling the **M52_DefineWatch** action routine and then making one or more calls to the following action routines:

- M52_WatchForCursorAt**
- M52_WatchForCursorNotAt**
- M52_WatchForIIOff**
- M52_WatchForChar**
- M52_WatchForNotChar**

Each of these actions encodes the requested watch command by appending some characters to the string variable **M52_WatchCommandString**. These characters are interpreted by 5250 Support for EASEL OS/2 EE at the time the watch commands are executed. (The contents of **M52_WatchCommandString** are managed by 5250 Support for EASEL OS/2 EE, so it is not necessary to know how the commands are encoded.)

The action routines **M52_Watch** and **M52_WatchAndWait** cause the current watch command sequence to be executed. Normally, one of these is called immediately after sending a keystroke that causes the host to transmit a new screen.

21.4.8.1 M52_DefineWatch

Syntax: **action M52_DefineWatch**

Input None

Variables:

Output **M52_WatchCommandString** (string)

Variables:

Remarks: **M52_DefineWatch** clears **M52_WatchCommandString** in preparation for defining a new sequence of watch commands to be executed by the **M52_Watch** or **M52_WatchAndWait** action routines.

M52_WatchCommandString is internal to the **M5250** module, and should not be referenced directly.

If a single behavior pattern defines the completion of screen transmission for all screens in the host application, **M52_DefineWatch** need only be called once. If the screens do not behave in the same fashion, **M52_DefineWatch** should be called each time a screen with changed behavior is expected.

Examples: See *Examples for:*

M52_WatchForCursorAt
M52_WatchForCursorNotAt
M52_WatchForIIOff
M52_WatchForChar
M52_WatchForNotChar

21.4.8.2 M52_WatchForCursorAt

Syntax: **action M52_WatchForCursorAt**

Input **M52_WatchRow** (integer)

Variables: **M52_WatchCol** (integer)

Output **M52_WatchCommandString** (string)

Variables:

Remarks: **M52_WatchForCursorAt** watches the screen until the cursor is located at the position specified by **M52_WatchRow** and **M52_WatchCol**.

M52_WatchForCursorAt does not get executed until a call is made to the **M52_Watch** or **M52_WatchAndWait** action routine.

M52_WatchCommandString is internal to the **M5250** module, and should not be referenced directly.

Example: action Sequence_1 is
 action M52_DefineWatch
 copy 22 to M52_WatchRow
 copy 54 to M52_WatchCol
 action M52_WatchForCursorAt

See Also: **M52_WatchForCursorNotAt**

21.4.8.3 M52_WatchForCursorNotAt

Syntax: **action M52_WatchForCursorNotAt**

Input **M52_WatchRow** (integer)

Variables: **M52_WatchCol** (integer)

Output **M52_WatchCommandString** (string)

Variables:

Remarks: **M52_WatchForCursorNotAt** watches the screen until the cursor is located at any position other than the one specified by **M52_WatchRow** and **M52_WatchCol**. In other words, it tests for the disappearance of the cursor from the specified position.

This command does not get executed until a call is made to the **M52_Watch** or **M52_WatchAndWait** action routine.

M52_WatchCommandString is internal to the **M5250** module, and should not be referenced directly.

Example: action Sequence_2 is
 action M52_DefineWatch
 copy 22 to M52_WatchRow
 copy 54 to M52_WatchCol
 action M52_WatchForCursorNotAt

See Also: **M52_WatchForCursorAt**

21.4.8.4 M52_WatchForIIOff

Syntax: **action M52_WatchForIIOff**

Input **M52_SettleTime** (integer)
Variables:

Output **M52_WatchCommandString** (string)
Variables:

Remarks: **M52_WatchForIIOff** watches the Operator Information Area (OIA) until the Input Inhibited indicator (II) is gone and remains gone for the period of time specified by **M52_SettleTime**. (The Input Inhibited indicator on a 5250 terminal is equivalent to the "X" in the OIA of a 3270 terminal.)

M52_SettleTime is in units of .01 seconds (the value of 100 is equivalent to 1 second). When the value of **M52_SettleTime** is zero, which it is initially, **M52_WatchForIIOff** completes immediately once the "II" is off. Specifying a non-zero value for **M52_SettleTime** is useful for host applications where the II flickers while the controller is writing data to the screen. Use of an appropriate value will ensure that the "II" is no longer flickering. If you do need to increase the value, it should be at least 20 (.20 seconds).

M52_WatchForIIOff does not get executed until a call is made to the **M52_Watch** or **M52_WatchAndWait** action routine.

M52_WatchCommandString is internal to the **M5250** module, and should not be referenced directly.

Example: action Sequence_3 is
 action M52_DefineWatch
 copy 20 to M52_SettleTime
 action M52_WatchForIIOff
 copy 15 to M52_WatchRow
 copy 56 to M52_WatchCol
 action M52_WatchForCursorAt

21.4.8.5 M52_WatchForChar

Syntax: **action M52_WatchForChar**

Input **M52_WatchRow** (integer)

Variables: **M52_WatchCol** (integer)

M52_WatchChar (string)

Output **M52_WatchCommandString** (string)

Variables:

Remarks: **M52_WatchForChar** watches the screen until the character specified by **M52_WatchChar** appears at the location specified by **M52_WatchRow** and **M52_WatchCol**. In other words, **M52_WatchForChar** tests for the appearance of a specific character at a specific location in the screen.

If **M52_WatchChar** contains more than one character, only the first character will be used. If you need to check for multiple characters on the screen, you can set up multiple **M52_WatchForChar** definitions.

M52_WatchForChar does not get executed until a call is made to the **M52_Watch** or **M52_WatchAndWait** action routine.

M52_WatchCommandString is internal to the **M5250** module, and should not be referenced directly.

Example: action Sequence_4 is
 action M52_DefineWatch
 copy 22 to M52_WatchRow
 copy 54 to M52_WatchCol
 copy "Q" to M52_WatchChar
 action M52_WatchForChar

See Also: **M52_WatchForNotChar**

21.4.8.6 M52_WatchForNotChar

Syntax: **action M52_WatchForNotChar**

Input **M52_WatchRow** (integer)
Variables: **M52_WatchCol** (integer)
 M52_WatchChar (string)

Output **M52_WatchCommandString** (string)
Variables:

Remarks: **M52_WatchForNotChar** watches the screen until any character other than the one specified by **M52_WatchChar** appears at the location specified by **M52_WatchRow** and **M52_WatchCol**. In other words, **M52_WatchForNotChar** tests for the disappearance of a specific character at a specific location in the screen.

If **M52_WatchChar** contains more than one character, only the first character will be used. If you need to check for multiple characters that are not on the screen, you can set up multiple **M52_WatchForNotChar** definitions.

M52_WatchForNotChar does not get executed until a call is made to the **M52_Watch** or **M52_WatchAndWait** action routine.

M52_WatchCommandString is internal to the **M5250** module, and should not be referenced directly.

Example: action Sequence_5 is
 action M52_DefineWatch
 copy 22 to M52_WatchRow
 copy 54 to M52_WatchCol
 copy V to M52_WatchChar
 action M52_WatchForNotChar

See Also: **M52_WatchForChar**

21.4.8.7 M52_Watch

Syntax: **action M52_Watch**

Input **M52_WatchCommandString** (string)

Variables: **M52_GiveUpTime** (integer)

Output None

Variables:

Remarks: **M52_Watch** executes the watch commands in **M52_WatchCommandString** while allowing EASEL OS/2 EE to continue to respond to other stimuli that do not pertain to the 5250 buffer (such as object selections and keyboard input), as long as those responses are within a guarded or resumable begin/end block.

The watch commands are executed one at a time, in the same order in which they were defined (via the **M52_WatchFor...** action routines), until either there are no more watch commands left, or a period of time specified by **M52_GiveUpTime** has elapsed since **M52_Watch** was called. **M52_GiveUpTime** is in units of .01 seconds (the value 100 is equivalent to 1 second). Its initial value is 3000 (30 seconds).

To detect the completion of **M52_Watch**, you must set up a guarded or resumable begin/end block with both of the following **response to line** statements, exactly as shown:

response to line "Watch done" from A5250
response to line "Watch gave up" from A5250

"Watch done" and "Watch gave up" are status messages from the A5250 local application. Normally, the message "Watch done" is received, unless **M52_GiveUpTime** elapses before all of the watch commands complete or an error has occurred.

We strongly recommend that you define the block as either **guarded** or **resumable**. If you want to restrict valid responses to be only those within the block, make the block **guarded**. If you want to have all other responses in outer blocks valid as well, make the block **resumable**. The begin/end block must have at least one **leave block** statement.

The **M52_Watch** action routine returns immediately after being called, but the watch commands continue executing until one of the above status messages is received from A5250. Calling any other **M5250** action routine while a watch is in progress interrupts and cancels the watch. No status message results from a canceled watch.

Example:

```

action M52_DefineWatch
copy 24 to M52_WatchRow      # Developer knows that the
copy 79 to M52_WatchCol     # next screen has an asterisk
copy "*" to M52_WatchChar   # at row 24, column 79.
action M52_WatchForChar
action M52_PressENTER        # Pressing ENTER causes the
                             # next screen to be display

# Watch for the next screen to be completed:
copy 2000 to M52_GiveUpTime  # Give up after 20 seconds.
action M52_Watch
begin guarded
    response to line "Watch done" from A5250
        copy "Mo_Report" to M52_TextRegionName
        action M52_ReadScreen
        leave block
    # If watch did not complete within 20 seconds,
    # have user try again.
    response to line "Watch gave up" from A5250
        change Display_Message to
            text "System error, try again"
        make Display_Message visible
        leave block
    response to Menu_keys
        # Respond to keys while waiting for next
        # screen. All other active responses must be
        # inside this block.
    ...
end

```

Errors: **M52_E_WATCH** Empty or invalid watch command string

See Also: **M52_WatchForChar**

21.4.8.8 M52_WatchAndWait

Syntax: **action M52_WatchAndWait**

Input Variables: **M52_WatchCommandString** (string)
M52_GiveUpTime (integer)

Output Variables: **M52_WatchGaveUp** (boolean)

Remarks: **M52_WatchAndWait** executes the watch commands in **M52_WatchCommandString** in the same way as **M52_Watch**, except that it does not allow EASEL OS/2 EE to respond to other stimuli while **M52_WatchAndWait** is running. Therefore, no begin/end blocks need to be defined.

The watch commands are executed one at a time, in the same order in which they were defined (via the **M52_WatchFor...** action routines), until either there are no more watch commands left, or a period of time specified by **GiveUpTime** has elapsed since **M52_WatchAndWait** was called.

M52_GiveUpTime is in units of .01 seconds (the value 100 is equivalent to 1 second). Its initial value is 3000 (30 seconds).

M52_WatchGaveUp is set to either **true** or **false**, depending on whether or not the **M52_GiveUpTime** has elapsed.

Example:

```
action M52_DefineWatch           # Developer knows next
copy 22 to M52_WatchRow         # screen will be complete
copy 6 to M52_WatchCol         # when cursor disappears
action M52_WatchForCursorNotAt  # from row 22, column 6.
copy 0 to M52_SettleTime
action M52_WatchForIIOff
action M52_PressENTER           # Pressing ENTER will cause the
                                # next screen to be displayed.

copy 2500 to M52_GiveUpTime     # Wait for next screen to
action M52_WatchAndWait         # be completed. If not
if (M52_WatchGaveUp) then       # completed in 25 seconds
    change Display_Message to   # have user try again.
    text "System error, try again"
    make Display_Message visible
else
    copy "Weekly_report" to     # If completed
    M52_TextRegionName         # within 25
    action M52_ReadScreen       # seconds,
                                # use it.
```

Errors: **M52_E_WATCH**

Empty or invalid watch command
string

See Also: **M52_Watch**

21.4.9 Communicating with the Host

Host applications may require user responses. These responses can include keyed data, menu selections, and special function keys. 5250 Support for EASEL OS/2 EE provides these capabilities through action routines that simulate various keyboard functions and combinations of functions.

21.4.9.1 M52_PressKey

Syntax: **action M52_PressKey**

Input **M52_KeyCode** (integer)

Variables: **M52_AutoWait** (boolean)

Output None

Variables:

Remarks: **M52_PressKey** simulates the pressing of a non-printing key that has special significance, such as TAB, ENTER, etc. Available key codes are shown here:

M52_ATTN	M52_FIELD_MINUS
M52_BACK_SPACE	M52_FIELD_PLUS
M52_BACK_TAB	M52_HELP
M52_CLEAR	M52_HEX
M52_CMD	M52_HOME
M52_DELETE	M52_INSERT
M52_DOWN_ARROW	M52_LEFT_ARROW
M52_DUP	M52_NEW_LINE
M52_END	M52_PRINT
M52_ENTER	M52_RECORD_BACKSPACE
M52_ERASE_EOF	M52_RIGHT_ARROW
M52_ERASE_INPUT	M52_ROLL_DOWN
M52_ERROR_RESET	M52_ROLL_UP
M52_FAST_DOWN_ARROW	M52_SYS_REQ
M52_FAST_LEFT_ARROW	M52_TAB
M52_FAST_RIGHT_ARROW	M52_TEST_REQUEST
M52_FAST_UP_ARROW	M52_UP_ARROW
M52_FIELD_EXIT	

Copy one of these string constants to **M52_KeyCode**.

If **M52_AutoWait** is set to **true**, the routine calls **M52_WaitForIOff** before returning, to provide automatic synchronization. The initial value for **M52_AutoWait** is **false**.

Example: response to NextField
 copy M52_TAB to M52_KeyCode
 action M52_PressKey

Errors: M52_E_SENDKEY Failed to send keystroke(s)

See Also: **M52_PressENTER**
 M52_PressCMDkey
 M52_TypeString
 M52_EnterString
 M52_WatchAndWait

21.4.9.2 M52_PressENTER

<i>Syntax:</i>	action M52_PressENTER	
<i>Input</i>	M52_AutoWait	(boolean)
<i>Variables:</i>		
<i>Output</i>	None	
<i>Variables:</i>		
<i>Remarks:</i>	M52_PressENTER simulates the pressing of the ENTER key on a 5250 terminal. This operation can also be performed by calling M52_PressKey with M52_KeyCode set to M52_ENTER. M52_PressENTER is provided for convenience, since the ENTER key is used frequently for many applications. It does not alter the value of M52_KeyCode. If M52_AutoWait is set to true, the routine calls M52_WaitForIOff before returning. The initial value for M52_AutoWait is false.	
<i>Example:</i>	<pre>response to RptSelect copy true to M52_AutoWait action M52_PressENTER if (M52_WaitGaveUp) then ... else ... end if</pre>	
<i>Errors:</i>	M52_E_SENDKEY	Failed to send keystroke(s)
<i>See Also:</i>	M52_PressKey. M52_PressCMDkey M52_TypeString M52_EnterString M52_WatchAndWait	

21.4.9.3 M52_PressCMDkey

Syntax: **action M52_PressCMDkey**

Input **M52_AutoWait** (boolean)
Variables: **M52_CMDkeyNumber** (integer)

Output None
Variables:

Remarks: **M52_PressCMDkey** simulates the pressing of a CMD key sequence on a 5250 terminal. **M52_CMDkeyNumber** specifies one of the special function keys on the terminal keyboard.

M52_CMDkeyNumber must be a number from 1 to 24. If it is greater than 24, the **M52_PressCMDkey** action is ignored.

If **M52_AutoWait** is set to **true**, the routine calls **M52_WaitForIIOff** before returning. The initial value for **M52_AutoWait** is **false**.

To send just the CMD key, use **M52_PressKey**.

Example: response to Another_Key
 copy 5 to M52_CMDkeyNumber # Press CMD5
 action M5 2_PressCMDkey

 response to More_Key
 copy 1 to M52_CMDkeyNumber # Press CMD1
 action M52_PressCMDkey

Errors: M52_E_SENDKEY Failed to send keystroke(s)

See Also: **M52_PressKey**
 M52_PressENTER
 M52_TypeString
 M52_EnterString
 M52_WatchAndWait

21.4.9.4 M52_TypeString

<i>Syntax:</i>	action M52_TypeString
<i>Input</i>	M52_Keystrokes (string)
<i>Variables:</i>	
<i>Output</i>	None
<i>Variables:</i>	
<i>Remarks:</i>	M52_TypeString simulates the typing of a string of data on a 5250 terminal keyboard. The string of data is specified by the variable M52_Keystrokes. M52_Keystrokes can contain only printing ASCII characters, including spaces and tabs. (An ASCII tab character “\t” translates to pressing the Tab key on a 5250 terminal.) Characters in the string are entered into the screen buffer starting at the current cursor location. If the string length equals or exceeds the field size, the consequences are identical to what would happen if the user entered the string directly from a 5250 terminal keyboard; in this case, it is dependent on the 5250 application.
<i>Example:</i>	enabled blue region Manufacturing size 400 40 at position 110 90 in Screen_1 white border parameter is "MF2599" move to 200 20 text center "MANUFACTURING" class RptSelect is Stat Defect Warehouse Manufacturing # MF2599 is typed in at the cursor position in the # screen buffer, followed by the ENTER key. response to RptSelect copy parameter of object to M52_Keystrokes action M52_TypeString action M52_PressENTER
<i>Errors:</i>	M52_E_SENDKEY Failed to send keystroke(s)
<i>See Also:</i>	M52_PressKey M52_PressENTER M52_PressCMDkey M52_EnterString

21.4.9.5 M52_EnterString

Syntax: **action M52_EnterString**

Input **M52_Keystrokes** (string)
Variables: **M52_AutoWait** (boolean)

Output None
Variables:

Remarks: **M52_EnterString** is similar to **M52_TypeString** except that the additional keystroke ENTER is sent. This is slightly faster than using **M52_TypeString** followed by **M52_PressENTER**.

If **M52_AutoWait** is set to **true**, the routine calls **M52_WaitForIIOff** before returning. The initial value for **M52_AutoWait** is **false**.

Example: response to RptSelect
 copy parameter of object to M52_Keystrokes
 action M52_EnterString

Errors: **M52_E_SENDKEY** Failed to send keystroke(s)

See Also: **M52_PressKey**
 M52_PressENTER
 M52_PressCMDkey
 M52_EnterString
 M52_WatchAndWait

21.4.10 Miscellaneous Action Routines

21.4.10.1 M52_WriteField

<i>Syntax:</i>	action M52_WriteField	
<i>Input</i>	M52_FieldNumber	(integer)
<i>Variables:</i>	M52_FieldText	(string)
	M52_FillFlag	(boolean)
<i>Output</i>	None	
<i>Variables:</i>		
<i>Remarks:</i>	M52_WriteField copies the string contained in M52_FieldText directly into the field specified by the variable M52_FieldNumber . If the field is longer than M52_FieldText , and if M52_FillFlag is set to true , then the field is padded on the right with blanks. If M52_FillFlag is set to false , the original field characters that were not overwritten by the input string will remain in the field. If M52_FieldText is longer than the field, trailing characters of M52_FieldText will be truncated.	
<i>Example:</i>	response to RptSelect action M52_ScanScreen copy 7 to M52_FieldNumber copy true to M52_FillFlag copy parameter of object to M52_FieldText action M52_WriteField action M52_PressENTER	
<i>Errors:</i>	M52_E_FLDNUM	Invalid field number
	M52_E_NOFLDS	Screen contains no fields or has not been scanned
	M52_E_PROTFLD	A protected field was specified

21.4.10.2 M52_CheckIndicators

Syntax: **action M52_CheckIndicators**

Input None

Variables:

Output **M52_InputInhibited** (boolean)
Variables: **M52_SystemAvailable** (boolean)
 M52_MessageWaiting (boolean)

Remarks: **M52_CheckIndicators** inquires about the state of the Input Inhibited indicator, the System Available indicator, and the Message Waiting indicator in the status line. The state is either **true** for on or **false** for off, and is returned in the variables **M52_InputInhibited**, **M52_SystemAvailable**, and **M52_MessageWaiting**. The initial value for each variable is **false**.

Example:

```
response to start
  copy "H" to M52_SessionName
  action M52_Init
  action M52_CheckIndicators
  if (M52_SystemAvailable) then
    send "Fully connected to host.\n" to errorlog
  else
    send "System is not available.\n" to errorlog
    action M52_Disconnect
    action M52_Stop
    exit
  end if

response to Ready
  action M52_CheckIndicators
  if (M52_InputInhibited) then
    change Display_Message to text
      "Input is inhibited. Please try again later.\n"
    make Display_Message visible
  end if

response to MessageInquiry
  action M52_CheckIndicators
  if (M52_MessageWaiting) then
    change Display_Message to
      text "There is a message waiting for you.\n"
    make Display_Message visible
  end if
```


21.4.10.3 M52_GetSessionInfo

Syntax: **action M52_GetSessionInfo**
Input **M52_SessionName** (string)
Variables:

Output **M52_SessionType** (integer)
Variables: **M52_NumRows** (integer)
 M52_NumCols (integer)

Remarks: Action **M52_GetSessionInfo** returns information about the session specified by **M52_SessionName**.
M52_SessionName can contain the short session name of any started session, whether or not it is currently connected. When **M52_GetSessionInfo** returns, **M52_SessionType** contains a value representing the type of session and **M52_NumRows** and **M52_NumCols** contain the dimensions of the session's screen buffer (these values are always 24 and 80, respectively, for 5250 host sessions).

Note that the value returned in **M52_NumRows** does not include the Operator Information Area. Also, the values returned in **M52_NumRows** and **M52_NumCols** are not meaningful for printer sessions.

The following integer constants must be used when testing the value of **M52_SessionType**:

Integer Constants	Meaning
M52_ST_5250_HOST	5250 host terminal session
M52_ST_5250_PRINTER	5250 printer session
M52_ST_3270_HOST	3270 host terminal session
M52_ST_3270_PRINTER	3270 printer session
M52_ST_PC	Session is local to the Programmable workstation
M52_ST_UNKNOWN	Session type cannot be determined

The following values are returned if an error (**M52_E_SESSNAME**) occurs:

Variable	Value
M52_SessionType	M52_ST_UNKNOWN
M52_NumRows	0
M52_NumCols	0

Example:

```

copy "D" to M52_SessionName
action M52_GetSessionInfo
If (M52_SessionType = M52_ST_3270_HOST) then
  action M52_Connect
  if (not M52_Error) then
    # Read the entire screen buffer of the active
    # connected session into a textual region.
    copy 1 to M52_TopRow
    copy M52_NumRows to M52_BottomRow
    copy 1 to M52_LeftCol
    copy M52_Numcols to M52_RightCol
    copy "CurrentScreen" to M52_RegionName
    clear CurrentScreen
    action M52_ReadColoredScreen
  end if
end if

```

Errors: M52_E_SESSNAME Invalid session name

21.4.10.4 M52_PrintScreen

Syntax: **action M52_PrintScreen**

Input None

Variables:

Output None

Variables:

Remarks: **M52_PrintScreen** prints the contents of the current 5250 screen. This routine is useful for developing and debugging EASEL OS/2 EE programs.

Example: response to line "print" from keyboard
 action M52_PrintScreen

21.5 Error Handling

21.5.1 Error Status Variables

The **M5250** module returns errors status information in three variables:

M52_Error	(boolean)
M52_ErrorCode	(integer)
M52_ErrorText	(string)

These variables are set by the **M5250** module each time an action routine is called.

M52_Error is set to **true** if an error occurred during execution of the last action routine. Otherwise, it is set to **false**. When **M52_Error** is **true**, the other two variables can be used to obtain more information about the error condition:

- **M52_ErrorCode** contains an error code number that denotes the particular error that occurred.
- **M52_ErrorText** contains an error message that also appears in the EASEL OS/2 EE errorlog file.

If **M52_Error** is **false**, **M52_ErrorCode** contains zero and **M52_ErrorText** contains a null string.

Each action routine described in this chapter lists the error conditions that can result when the action routine is called, showing the error code and the associated error message text. The error code value is always represented by an integer constant whose name begins with **M52_E_**. (See 21.7, "Reserved Constants" on page 21-70). Always use these constants when testing the value of **M52_ErrorCode**.

In general, it is not necessary to have statements in your program that check for all possible errors after every action routine call. Many of the error conditions are caused by coding mistakes, as in the following example, and should never occur in a fully debugged program.

Suppose the following EASEL OS/2 EE statements are executed:

```
copy -3 to M52_Row    # M52_Row should not be less than 1
action M52_ReadLine   # ...this will cause an error
```

After **M52_ReadLine** returns, the error variables are set as follows:

M52_Error	=	true
M52_ErrorCode	=	M52_E_LINENUM
M52_ErrorText	=	"Invalid line number"

where: **M52_E_LINENUM** is an integer constant whose value is the error code corresponding to this error condition. The errorlog contains the line:

5250 Error: Invalid line number

21.5.2 Error Messages (Cross Reference)

In the following, the error message in the EASEL OS/2 EE errorlog file is listed along with the constant that corresponds to this error condition, and a brief explanation.

21.5.2.1 Cannot Initialize

Message: **5250 Error: Cannot initialize**

Constant: **M52_E_CANTINIT**

Remarks: **M52_Init** could not initialize. This error indicates that required hardware or software is not installed (for example, the IBM Enhanced 5250 Emulation Adapter is installed, but the emulator software is not resident).

See: **M52_Init**

21.5.2.2 Syntax Error in Command Line Sent from EASEL OS/2 EE

Message: **5250 Error: Syntax error in command line sent from EASEL OS/2 EE**

Constant: **M52_E_COMMAND**

Remarks: A command was sent from the **M5250** module that could not be deciphered by the A5250 local application. This error occurs if there is a version mismatch between **M5250** and A5250. Make sure that 5250 Support for EASEL OS/2 EE has been properly installed and that there are no out-of-date versions on your system.

21.5.2.3 Invalid Field Number

Message: **5250 Error: Invalid field number**

Constant: **M52_E_FLDNUM**

Remarks: The value of **M52_FieldNumber** was not a number between 1 and **M52_FieldCount** inclusive.

See: **M52_ReadField**
 M52_GetFieldLength
 M52_GetFieldAttributes
 M52_GetFieldPosition
 M52_WriteField

21.5.2.4 Session Is Already in Use

Message: **5250 Error: Session is already in use**

Constant: **M52_E_INUSE**

Remarks: The name specified by **M52_SessionName** was valid but it referred to a session already connected by another process.

See: **M52_Init**
 M52_Connect
 M52_GetSessionInfo

21.5.2.5 Invalid Line Number

Message: **5250 Error: Invalid line number**

Constant: **M52_E_LINENUM**

Remarks: The specified line number was not a value between 1 and 24 inclusive.

See: **M52_ReadLine**
 M52_FindLastNonBlankLine

21.5.2.6 Screen Contains No Fields or Has Not Been Scanned

Message: **5250 Error: Screen contains no fields or has not been scanned**

Constant: **M52_E_NOFLDS**

Remarks: An action routine that uses **M52_FieldNumber** was called when the value of **M52_FieldCount** was zero.

See: **M52_ReadField**
 M52_GetFieldLength
 M52_GetFieldAttributes
 M52_GetFieldNumber
 M52_GetFieldPosition
 M52_WriteField

21.5.2.7 Not Connected to an Active Session

Message: **5250 Error: Not connected to an active session**

Constant: **M52_E_NOTCONN**

Remarks: An **M5250** action routine was called that required an active session, and there was none.

See: **M52_Disconnect**

21.5.2.8 Not Initialized - Command Was Ignored

Message: **5250 Error: Not initialized - command was ignored**

Constant: **M52_E_NOTINIT**

Remarks: **M52_Init** was not the first **M5250** action routine called. All other action routines cause this error message until **M52_Init** gets called.

21.5.2.9 Attempt to Write Protected Field

Message: **5250 Error: Attempt to write protected field**

Constant: **M52_E_PROTFLD**

Remarks: Action **M52_WriteField** was called with a protected field specified.

See: **M52_WriteField**

21.5.2.10 (Row, Column) Is Not on Screen

Message: **5250 Error: (Row, Column) is not on screen**

Constant: **M52_E_ROWCOL**

Remarks: The input variables **M52_Row** and **M52_Column** specified a character position that was not within the 24-line by 80-column screen buffer.

See: **M52_GetFieldNumber**

21.5.2.11 Failed to Send Keystrokes

Message: **5250 Error: Failed to send keystrokes**

Constant: **M52_E_SENDKEY**

Remarks: An attempt to send a keystroke to the controller failed.

See: **M52_PressKey**
 M52_PressEnter
 M52_PressCMDKey
 M52_TypeString
 M52_EnterString

21.5.2.12 Invalid Screen Area Limits

Message: **5250 Error: Invalid screen area limits**

Constant: **M52_E_SCRNAREA**

Remarks: The values of **M52_TopRow**, **M52_BottomRow**,
 M52_LeftCol, and **M52_RightCol** did not specify an area
 that resides entirely within the 24-line by 80-column
 screen buffer.

See: **M52_ReadScreen**
 M52_ReadColoredScreen

21.5.2.13 Invalid Session Name

Message: **5250 Error: Invalid session name**

Constant: **M52_E_SESSNAME**

Remarks: The value of **M52_SessionName** was not a valid session
 name. A valid name must be one of the 5250 host session
 short names configured by CM, and it must have been
 started by CM.

See: **M52_Init**
 M52_Connect
 M52_GetSessionInfo

21.5.2.14 Empty or Invalid M52_WatchCommandString

Message: **5250 Error: Empty or invalid M52_WatchCommandString**

Constant: **M52_E_WATCH**

Remarks: No **M52_WatchFor...** action routines were called before calling **M52_Watch** or **M52_WatchAndWait**.

See: **M52_Watch**
 M52_WatchAndWait

21.6 Reserved Variables (Cross-Reference)

The following reserved variables are used in the **M5250** module. They are all global variables. Be sure that your program does not contain any of these identifiers, except when referencing the **M5250** module.

The initial values are set by the action routine **M52_Init**.

Global Variable Name	Type	Initial Value	Action Routine References (O = output, I = input)
M52_AutoWait	boolean	false	M52_PressKey (I) M52_PressENTER (I) M52_PressCMDkey (I) M52_EnterString (I)
M52_BottomRow	integer	24	M52_ReadScreen (I) M52_ReadColoredScreen (I)
M52_CMDkeyNumber	integer	1	M52_PressCMDkey (I)
M52_Column	integer	1	M52_GetCursorPosition (O) M52_GetFieldNumber (I) M52_GetFieldPosition (O)
M52_Error	boolean	false	(all action routines) (O)
M52_ErrorCode	integer	0	(all action routines) (O)
M52_ErrorText	string	""	(all action routines) (O)
M52_FieldCount	integer	0	M52_ScanScreen (O)
M52_FieldLength	integer	0	M52_GetFieldLength (O)
M52_FieldNumber	integer	0	M52_ReadField (I) M52_GetFieldLength (I) M52_GetFieldAttributes (I) M52_GetFieldNumber (O) M52_GetFieldPosition (I) M52_WriteField (I)
M52_FieldText	string	""	M52_ReadField (O) M52_WriteField (I)
M52_FieldType	integer	M52_FT_ALPHANUMERIC	M52_GetFieldAttributes (O)
M52_FillFlag	boolean	false	M52_WriteField (I)
M52_GiveUpTime	integer	3000	M52_InformWhenIOff (I) M52_WaitForIOff (I) M52_Watch (I) M52_WatchAndWait (I)
M52_InputInhibited	boolean	false	M52_CheckIndicators (O)
M52_Intensified	boolean	false	M52_GetFieldAttributes (O)

Global Variable Name	Type	Initial Value	Action Routine References (O = output, I = input)
M52_KeyCode	integer	TAB	M52_PressKey (I)
M52_Keystrokes	string	" "	M52_TypeString (I) M52_EnterString (I)
M52_LeftCol	integer	1	M52_ReadScreen (I) M52_ReadColoredScreen (I)
M52_LineText	string	" "	M52_ReadLine (O)
M52_MessageWaiting	boolean	false	M52_CheckIndicators (O)
M52_Modified	boolean	false	M52_GetFieldAttributes (O)
M52_NonDisplay	boolean	false	M52_GetFieldAttributes (O)
M52_NumCols	integer	0	M52_GetSessionInfo (O)
M52_NumRows	integer	0	M52_GetSessionInfo (O)
M52_Protected	boolean	false	M52_GetFieldAttributes (O)
M52_RightCol	integer	80	M52_ReadScreen (I) M52_ReadColoredScreen (I)
M52_Row	integer	1	M52_GetCursorPosition (O) M52_GetFieldNumber (I) M52_GetFieldPosition (O) M52_ReadLine (I) M52_FindLastNonBlankLine (O)
M52_SessionName	string	"*"	M52_Init (I) M52_Connect (I)
M52_SessionType	integer	M52_ST_UNKNOWN	M52_GetSessionInfo (O)
M52_SessionNumber	integer	1	M52_SelectSession (archaic)
M52_SettleTime	integer	0	M52_WatchForIIOff (I)
M52_StartLine	integer	24	M52_FindLastNonBlankLine (I)
M52_SystemAvailable	boolean	false	M52_CheckIndicators (O)
M52_TextRegionName	string	"TR5250"	M52_ReadScreen (I) M52_ReadColoredScreen (I)
M52_TopRow	integer	1	M52_ReadScreen (I) M52_ReadColoredScreen (I)
M52_WaitGaveUp	boolean	false	M52_WaitForIIOff (O)
M52_WatchChar	string	" "	M52_WatchForChar (I) M52_WatchForNotChar (I)
M52_WatchCol	integer	1	M52_WatchForCursorAt (I) M52_WatchForCursorNotAt (I) M52_WatchForChar (I) M52_WatchForNotChar (I)

Global Variable Name	Type	Initial Value	Action Routine References (O = output, I = input)
M52_WatchCommandString	string	" "	M52_DefineWatch (O) M52_WatchForCursorAt (O) M52_WatchForCursorNotAt (O) M52_WatchForIIOff (O) M52_WatchForChar (O) M52_WatchForNotChar (O) M52_Watch (I) M52_WatchAndWait (I)
M52_WatchGaveUp	boolean	false	M52_WatchAndWait (O)
M52_WatchRow	integer	1	M52_WatchForCursorAt (I) M52_WatchForCursorNotAt (I) M52_WatchForChar (I) M52_WatchForNotChar (I)

21.7 Reserved Constants

The following reserved constants are used in the **M5250** module. Be sure that your program does not contain any of these identifiers, except when referencing the **M5250** module.

M52_ATTN	M52_FIELD_EXIT
M52_BACK_SPACE	M52_FIELD_MINUS
M52_BACK_TAB	M52_FIELD_PLUS
M52_CLEAR	M52_FT_ALPHANUMERIC
M52_CMD	M52_FT_ALPHA_ONLY
M52_DELETE	M52_FT_NUMERIC_SHIFT
M52_DOWN_ARROW	M52_FT_NUMERIC_ONLY
M52_DUP	M52_FT_RESERVED
M52_E_CANTINIT	M52_FT_DIGITS_ONLY
M52_E_COMMAND	M52_FT_MAG_STRIPE
M52_E_FLDNUM	M52_FT_SIGNED_NUMERIC
M52_E_INUSE	M52_HELP
M52_E_LINENUM	M52_HEX
M52_E_NOFLDS	M52_HOME
M52_E_NOTCONN	M52_INSERT
M52_E_NOTINIT	M52_LEFT_ARROW
M52_E_PROTFLD	M52_NEW_LINE
M52_E_ROWCOL	M52_PRINT
M52_E_SCRNAREA	M52_RECORD_BACKSPACE
M52_E_SENDKEY	M52_RIGHT_ARROW
M52_E_SESSNAME	M52_ROLL_DOWN
M52_E_SESSNO (archaic)	M52_ROLL_UP
M52_E_WATCH	M52_ST_3270_HOST
M52_END	M52_ST_3270_PRINTER
M52_ENTER	M52_ST_5250_HOST
M52_ERASE_EOF	M52_ST_5250_PRINTER
M52_ERASE_INPUT	M52_ST_PC
M52_ERROR_RESET	M52_ST_UNKNOWN
M52_FAST_DOWN_ARROW	M52_SYS_REQ
M52_FAST_LEFT_ARROW	M52_TAB
M52_FAST_RIGHT_ARROW	M52_TEST_REQUEST
M52_FAST_UP_ARROW	M52_UP_ARROW

21.8 Sample Program

Sample 5250 Screen "A":

MIN01001	ABC MANUFACTURING COMPANY
REPORT SELECTION	
Report Name	ID Number
-----	-----
Production Statistics	PS4471
Defect Reporting	DF1678
Warehouse Inventory	WI3672
Manufacturing Lead Time	MF2599
Enter Report ID Number: _____	

Sample 5250 Screen "B":

DF167801

Divn. : Acme ABC MANUFACTURING COMPANY
Report No: DF1678 DEFECTIVE PARTS REPORTING SYSTEM

Part Description	JAN	FEB	MAR	APR	MAY
3061 Cam Shaft	12	11	9	10	14
Bearing Ring	11	9	11	8	7
Time Gear - Large	11	19	14	16	15
Time Gear - Small	12	15	16	18	14
Cam Shaft Dowel Pin	15	16	13	16	15
Cam Shaft Lock Nut	9	11	12	12	13
Valve Bushing	12	9	7	9	19
	---	---	---	---	---
TOTAL	82	90	82	89	97

- More -

Cmd1 = NEXT PAGE

CMD7 = RETURN TO MENU

21.8.1 Sample Program Using 5250 Support for EASEL OS/2 EE

```
screen size device units      # Use resolution of device
font "system"
module M5250

include "accel.inc"          # Include accelerator keys for action bar
include "message.inc"        # Include message box routines

boolean variable More is false

string variable LineChosen
                PartName
                ReplyIs
                MessageIs

# ACTION BAR DEFINITION

action bar ActionBar is
    pulldown File text "~File"
        disabled choice Open text "~Open..."
        disabled choice New text "~New"
        separator
        disabled choice Save text "~Save"
        disabled choice SaveAs text "Save ~As..."
        separator
        choice Exit text "E~xit\tF3"
            accelerator is KEY_F3
    end pulldown
    pulldown ReportList text "~Reports"
        choice Prodstat text "~Production Stats"
            parameter is "PS4471"          # This is the ID No. (shown in
                                           # Sample Screen A)
        choice Defect text "~Defect Reports"
            parameter is "DF1678"
        choice Warehouse text "~Warehouse Inventory"
            parameter is "WI3672"
        choice Manufacturing text "~Manufacturing"
            parameter is "MF2599"
    end pulldown
end action bar

# OBJECT DEFINITIONS

white primary region Primary size 500 350 at position 75 75
    title bar "ABC Manufacturing Company"
    system menu
    action bar ActionBar
    minimize button
    maximize button
```

```

# Establish a class of items that includes all of the Report choices:

item class RptSelect is Prodstat Defect Warehouse Manufacturing

# Create a textual region to show the selected report:

invisible blue textual region Text window size 60 columns 17 lines
  at position 120 5 in desktop
  title bar "Requested Report"
  system menu
  vertical scroll bar
  border
  font "asc814"

# ACTION SUBROUTINES

# Copy current screen into Text textual region, and check if there are mo
# screens to come:

action ReadReportPage is
  copy "Text" to M52_TextRegionName
  action M52_ReadScreen                                # Appends screen data to Text
  copy 22 to M52_Row
  action M52_ReadLine
  switch M52_LineText is
    case char "- More -" is
      copy true to More
    default is
      copy false to More
  end switch

# At first page of report, read all pages into Text textual region:

action GetPages is
  action ReadReportPage
  while (More) loop
    copy 1 to M52_CMDkeyNumber      # Get next page if "- More -"
    action M52_PressCMDkey          # is displayed
    action M52_WaitForIIOff
    if M52_WaitGaveUp then
      copy false to More
    else
      action ReadReportPage
    end if
  end loop
  copy 7 to M52_CMDkeyNumber        # Return to menu screen
  action M52_PressCMDkey
  action M52_WaitForIIOff

```

```

# Log off from the AS/400:

action Logoff is
    # Code to log off from the AS/400 should be placed here.

# RESPONSE DEFINITIONS

response to start
    copy "A" to M52_SessionName
    action M52_Init
    # Code to log on to the AS/400 should be placed here

response to item RptSelect                # Class of reports
    copy parameter to M52_Keystrokes
    action M52_TypeString
    action M52_PressENTER
    action M52_WaitForIIOff
    if M52_WaitGaveUp then                # Put up message box with problem
        copy "Could not get report " Keystrokes "." to MessageIs
        copy (ReplyToMessage("System Problems", MessageIs,
            MessageOK, MessageOK, MessageIconHand)) to ReplyIs
        make Text invisible
    else
        action GetPages
        make Text visible
    end if

response to item Exit
    action Logoff
    action M52_Disconnect
    action M52_Stop
    exit

response to Primary on close
    action Logoff
    action M52_Disconnect
    action M52_Stop
    exit

```

Chapter 22. Printing and Plotting

22.1 Printing and Plotting EASEL OS/2 EE Regions

22.1.1 About Output Devices

EASEL OS/2 EE supports hardcopy devices through the hardcopy interface provided by the OS/2 Presentation Manager (PM). Therefore, it supports devices that have a full PM-compatible driver. Consult the manufacturer of your hardcopy device to find out if it has PM support.

The quality and appearance of output from hardcopy devices will vary depending on their functional capabilities and the extent of their support under PM. These dependencies include, among others:

- Colors and color mappings
- Size of output
- Font sizing and styles.

22.1.2 Overview of Hardcopy Support

The following describes the hardcopy support provided through PM:

- Support is device independent (subject to the limits of the device and its PM driver).
- All hardcopy devices that have PM drivers are supported.
- The full resolution of the output device is used.
- Objects appear on the page the same relative size as on the screen.
- The PM Control Panel controls portrait/landscape orientation.
- The plot command works in conjunction with the OS/2 spooler.
- The plot command may specify a particular printer or use the default printer.

The following restrictions apply:

- The OS/2 spooler must be active.
- Only EASEL OS/2 EE graphical and textual regions and keys can be output. PM objects, such as dialog boxes, dialog regions, dialog controls, action bars, and so on, cannot be output. Image regions cannot be output.
- Any region with frame controls will appear on the output page with a 1-pixel wide border instead of the frame controls.
- Only color mapping controlled by the PM Control Panel is supported.

22.1.3 Using the Plot Action Statement

The plot action statement allows you to use any output device that has a PM driver.

Model

```
plot { OBJECT_NAME | screen } [OPTION_STRING]
```

As shown in the model, you can print either a selected object or the entire screen. For example:

```
response to Key_PtSc
  plot screen
```

or:

```
response to Print_Key
  plot Graph "printer PRINTER2"
```

When an object is plotted, it is drawn on the output page with the same relative position and size as it is displayed on the monitor. Specify **screen** only when you are creating a fullscreen EASEL OS/2 EE interface.

22.1.3.1 OPTION_STRING Parameters

You can use the following OPTION_STRING parameters in the plot Action Statement, in any order:

keyword	Description
printer PRINTER_NAME	Places a job in the printer queue for the selected printer. If more than one queue is assigned to PRINTER_NAME, the default queue is used. Default: default printer.
nobackground	Causes all regions to be drawn without background. This is most useful for hardcopy devices such as plotters and printers with limited color capability. Some of these devices might output the background in a dark color that would obscure the foreground. Default: background is drawn.
noctext	Causes colored textual regions to be drawn as if they are non-colored. Text is drawn in foreground color of the region.

Default: colored text is drawn using the background and foreground colors of individual characters.

invertbw

Causes white to be drawn as black and black as white.

Default: white and black are not inverted.

22.1.3.2 Error Messages

The following error messages may appear in the errorlog after the **plot** statement is executed:

error in plotting: error parsing plot options

One of the options in the **plot** action statement is incorrect.

error in plotting: can't find default printer name

A default printer has not been defined. Use the PM Control Panel to define the default printer.

error in plotting: can't find queue and device driver for PRINTER_NAME

No printer queue or device driver is associated with PRINTER_NAME. Use the PM Control Panel to do this.

error in plotting: can't open device context

The OS/2 spooler is not configured properly. Use the PM Control Panel to check the configuration.

error in plotting: can't create presentation space

A system error has occurred or there is not enough system memory.

22.2 Printing ASCII Files Within EASEL OS/2 EE

The ESLPRS.EXE and ESLPR.EX utilities are provided to print an ASCII file from within an EASEL OS/2 EE program. ESLPRS.EXE is designed to operate with the PM Print Spooler enabled. This is the default for OS/2 and is the recommended method of printing.

The following information applies also to ESLPR.EXE. ESLPR.EXE is designed to operate when the spooler is not enabled. ESLPR.EXE initializes the printer before printing and performs device status inquiries as it is printing.

22.2.1 Using the Printing Utilities

The following model shows how to use these utilities.

Model (Partial Code)

```
application APPLICATION_NAME
:

response to start
    start local APPLICATION_NAME "ESLPRS.EXE"
    :

response to ...
    send "file FILENAME [PARAMETERS]" to APPLICATION_NAME
    :
```

22.2.2 Optional Parameters

APPLICATION_NAME is the identifier representing the logical name you assign to the application "ESLPRS.EXE."

FILENAME is a string value representing the file name and extension of the file to be printed. The full path can be specified.

PARAMETERS is one or more of the following optional keywords, separated by a space, that control the printer:

Keyword	Description
lpt1	Printer port #1 (<i>Default</i>)
lpt2	Printer port #2
lpt3	Printer port #3
header	Prints a header on each page.
pagesize NUM	Specifies the number of lines per page. If header has been specified, include an extra 3 lines in the pagesize . The default is pagesize 58, which is 58 lines per page (55 lines if header was specified).

The ASCII file can contain form feed characters (^L), causing the printer to skip to the top of the next page. If the line containing the form feed character also contains other characters, those characters will be printed on the current page before the form feed is executed. If the form feed is on the last line of the page, the following printed page will be blank.

ESLPRS.EXE tells EASEL OS/2 EE if an error occurs, and when it is ready to print another file. ESLPRS.EXE sends "DONE" to EASEL OS/2 EE when it has successfully finished sending the file to the printer.

22.2.3 Example

```
application PrintFile
:
response to start
    start local PrintFile "eslprs.exe"

response to PrintKey
    send "file figures.txt pagesize 40" to PrintFile

# Always provide a response to handle errors:
response to line "ERROR:" from PrintFile
:
# Provide a response to handle completion:
response to line "DONE" from PrintFile
:
```

22.2.4 Error Messages

ERROR: Can't open file.

Specified file does not exist.

ERROR: A file name is required.

No file name has been specified.

ERROR: Too few lines per page specified.

You may have specified a header, but did not add three extra lines to **pagesize**.

ERROR: Unrecognized parameter.

You may have misspelled one of the keywords in the parameter.

ERROR: Unexpected End-of-File encountered.

An end-of-file character (^Z) was encountered in the file.

Chapter 23. Environment Files

23.1 Overview

The EASEL OS/2 EE environment files are used for configuration (**CONFIG.ESL**), compilation (**COMPILE.CMD**), and execution (**EASEL.CMD**) of EASEL OS/2 EE graphical user interface applications.

Note: For compilation or execution, either the default configuration file (**CONFIG.ESL**) can be used or an application specific configuration file can be specified (see 23.2.1, “Command Line Options” on page 23-3 and 23.4.1, “Command Line Options” on page 23-9.)

In addition to the environment files, EASEL OS/2 EE also provides the following utilities:

ESLCUA.EXE	CUA user interface definition facility (Layout/CUA)
AUDIO.EBI	Audio capture and playback utility
FIELDS.EXE	3270 screen buffer analysis utility
FIELDS52.EXE	5250 screen buffer analysis utility
ESLPRS.EXE	ASCII text file printing utility for OS/2 Print Manager
ESLPR.EXE	ASCII text file printing utility
TSCAL.CMD	Touchscreen calibration utility

23.2 COMPILE.CMD OS/2 Command File

The COMPILE.CMD file is used to specify the commands and arguments that are executed when compiling an EASEL OS/2 EE program.

When you compile an EASEL OS/2 EE program, EASEL OS/2 EE transforms the .EAL program statements into a binary representation contained in an OS/2 file, whose name will have the extension .EBI. The .EBI file is the executable interface that can be executed by **EASEL.CMD**. The compilation will also produce an errorlog file.

The COMPILE.CMD file is executed when you enter:

```
compile [PROGRAM_NAME] [COMMAND_LINE_OPTION]...
```

at an OS/2 command prompt. For example, you can specify:

```
compile myprog -m 150000 -e errors
```

The command line options below can be specified when executing the COMPILE.CMD file for a temporary change to the program execution. To have the COMPILE.CMD file default to using particular command line options, the EPARSE line in the COMPILE.CMD file can be modified to include any of the command line options below. These additional default command line options should be inserted between EPARSE and %1. EPARSE is the program that the COMPILE.CMD file executes to transform EASEL OS/2 EE program statements into an EASEL OS/2 EE executable interface. The command line options added to the EPARSE line in the COMPILE.CMD file can be temporarily changed by specifying the new value when executing the COMPILE.CMD file.

23.2.1 Command Line Options

-b PROGRAM_NAME.ebi

PROGRAM_NAME is the name EASEL OS/2 EE should use as the output file name. It will contain the binary version of the program after a successful compile. If this option is not specified, the .EAL name specified will be used and the .EAL extension will be replaced with .EBI.

Initial value: not specified

-debug

If specified, must be in both COMPILE.CMD and EASEL.CMD files.

Causes the line numbers from the .EAL files to be placed in the errorlog when traces and errors are generated during runtime.

Initial value: not specified

-e FILENAME

Sends error messages to FILENAME.

Initial value: not specified

-f FILENAME

Defines which configuration file EASEL OS/2 EE is to use.

Initial value: not specified. By default, the filename is CONFIG.ESL, and it is searched for in the directories in the OS/2 PATH specification.

-fullscreen

Specifies that the application is fullscreen. It must be used if the application does not run in a window. Fullscreen applications do not allow the definition of a primary region. Running a fullscreen application causes the EGA/VGA color palette to be set to the fullscreen color set. Specifying the fullscreen flag also affects the way selectability works; in a fullscreen application, regions are disabled by default, and a region's selectability does not affect the selectability of its children.

-g SIZE

Specifies size, in bytes, of allocated global memory.

Initial value: not specified. This overrides the value of GLBSIZE specified in the configuration file CONFIG.ESL.

-lx

Places the EASEL OS/2 EE program text into the errorlog as the program is compiled. When using this option, the **-e** option should also be used. (Note that the "l" specified is the alphabetic character, not the numeric character "1".)

Initial value: not specified

-m SIZE

Specifies initial size, in bytes, of EASEL OS/2 EE program memory.

Initial value: not specified. This overrides the value of PRGSIZE specified in the configuration file CONFIG.ESL.

23.3 CONFIG.ESL File

The CONFIG.ESL file is used to specify the parameters that configure EASEL OS/2 EE for your computer.

You can make a permanent change to any option by entering the options and values in the CONFIG.ESL file. For example:

```
ESLERR=C:\ease1-ee\eslerr.txt
EXETYPE=.exe
FONTPATH=C:\ease1-ee\d11
HLDELAY=8000
MODPATH=C:\ease1-ee\modules
BEEPFREQ=1250
BOOPFREQ=250
TEXTYPE=.txt
```

You can make a temporary change to any option by entering `set OPTION=VALUE` at the OS/2 command prompt, which overrides the value specified in the CONFIG.ESL file. The SET command affects EASEL OS/2 EE executions until the option is reset or until the system is rebooted. For example:

```
set BEEPFREQ=1000
set BOOPFREQ=200
```

You can also make a permanent change to any option by entering the appropriate SET command in the EASEL.CMD file or COMPILE.CMD file.

Options that have path values are set by the installation procedure and should not be changed.

Do not remove any lines from the CONFIG.ESL file.

23.3.1 Configuration Options

BEEPFREQ = NUMBER

Tone frequency, in Hertz, of the pointing device sound for correct touches. The higher the number, the higher the frequency.

Initial value: 1250

BOOPFREQ = NUMBER

Tone frequency, in Hertz, of the pointing device sound for incorrect touches. The higher the number, the higher the frequency.

Initial value: 250

CODEPAGE = NUMBER

Code page to use when compiling and running the EASEL OS/2 EE program. This is set by the installation program. Available values are: 850 (recommended), 437, 860, 863, 865.

Initial value: 850

ESLERR = FILENAME

Name of source file for error messages added to the errorlog.

Initial value: c:\easel-ee\eslerr.txt

EXETYPE = EXTENSION

Default file extension for executables.

Initial value: .exe

FontCNT = NUMBER

Number of fonts that can be loaded into memory at any one time by an EASEL OS/2 EE program.

Initial value: 20

FontPATH = PATH

Directories to search for font files.

Initial value: c:\easel-ee\dll

CAUTION: Do not place a copy of ESLFONTS.FON file in your current directory.

GLBMAX = NUMBER

Maximum size of EASEL OS/2 EE global memory, in kilobytes. This must be larger than or equal to GLBSIZE.

Initial value: 50

GLBSIZE = NUMBER

Size of EASEL OS/2 EE global memory, in kilobytes. This value may be overridden by the value specified after the command line option **-g** (if specified) in the .CMD files.

Initial value: 10

HLDELAY = NUMBER

Highlight delay parameter. A value between 0 and 32000 can be specified. The larger the value, the longer EASEL OS/2 EE will flash a highlight.

Initial value: 8000

INCPATH = PATH

Directories to search for "include" files.

Initial value: c:\easel-ee

INCTYPE = EXTENSION

Default file extension used for "include" files.

Initial value: .inc

KBDTABWID = NUMBER

Amount of spaces to translate a tab character into when typed from the keyboard. If this value is 0, tabs are not converted to spaces in application input.

Initial value: **0**. This leaves the tab character instead of translating it to spaces.

LOCALDLL = NAME

Name of DLL for supporting local applications. This value is set by the installation procedure and should not be changed.

Initial value: **local**

MODPATH = PATH

Directories to search for modules.

Initial value: **c:\easel-ee\modules**

POINTER = POINTER

EASEL OS/2 EE's default pointing device. Value can be **mouse** or **touchscreen**.

Initial value: **mouse**

PRGMAX = NUMBER

Maximum size of EASEL OS/2 EE program memory, in kilobytes. Must be larger than or equal to PRGSIZE.

Initial value: **500**

PRGSIZE = NUMBER

Initial size of EASEL OS/2 EE program memory, in kilobytes. This value may be overridden by the value specified after the command line option **-m**, if specified.

Initial value: **300**

REMOTE = STRING

Remote application settings. This string is inserted before the settings specified in the **start remote** action statement.

Initial value: **1200 nopar stopbits 1 tandem**

REMOTEDLL = NAME

Name of DLL for supporting remote application. This value is set by the installation process and should not be changed.

Initial value: **ESLACDI**

SLEEPMAX = NUMBER

Maximum number of seconds allowed for a **wait** action statement.

Initial value: **60**

TABWIDTH = NUMBER

Standard number of columns per tab, during application input. If this value is 0, tabs are not converted to spaces in application input.

Initial value: **8**

TEXTPATH = PATH

Directories to search when reading files into a textual region.

Initial value: not specified. Only current directory is searched.

TEXTTYPE = EXTENSION

Default extension for files read into and written to from textual regions.

Initial value: not specified. Looks for a file with no extension.

XMAX and YMAX

Set in the CONFIG.ESL file during installation. Do not modify.

23.4 EASEL.CMD OS/2 Command File

The EASEL.CMD file is used to specify the commands and arguments that are executed when executing an EASEL OS/2 EE program.

To execute an EASEL OS/2 EE program, it must have been previously compiled, which can be done using **COMPILE.CMD**. The program is the executed by entering:

```
easel [PROGRAM_NAME] [COMMAND_LINE_OPTION]...
```

at an OS/2 command prompt. For example you can specify:

```
easel myprog -m 150000 -e errors
```

The command line options below can be specified when executing the EASEL.CMD file for a temporary change to the program execution. To have the EASEL.CMD file default to using particular command line options, the ESLRUN line in the EASEL.CMD file can be modified to include any of the command line options below. These additional default command line options should be inserted between ESLRUN and %1. EPARSE is the program that the EASEL.CMD file executes to run an EASEL OS/2 EE executable interface. The command line options added to the ESLRUN line in the EASEL.CMD file can be temporarily changed by specifying the new value when executing the EASEL.CMD file.

23.4.1 Command Line Options

-apio

Messages tracing communication to and from applications are placed in the errorlog file. When using this option, the **-e** option should also be used. Used for debugging.

Initial value: not specified

-d VAR_NAME INITIAL_VALUE

Initializes EASEL OS/2 EE scalar (nonarray) integer, float, boolean, and string variables. The variable must be declared in the program being started. The initialization occurs once, when EASEL OS/2 EE is started, and it overrides program initialization values. This option does not affect subsequent program changes, except that global variables retain their values. To initialize a string variable with a value containing more than one word, enclose the entire initialization value in double quotes ("). EASEL OS/2 EE attempts to convert the initial value to match the type of the variable. It is an error if (1) EASEL OS/2 EE determines that the types do not match, (2) there is no initial value, (3) the variable is undefined, (4) the variable is not one of the types listed above, or (5) the variable is an array.

Initial value: not specified

-debug

If specified, must be in both COMPILE.CMD and EASEL.CMD files.
Causes the line numbers from the .EAL files to be placed in the errorlog when traces and errors are generated during runtime. When using this option, the **-e** option should also be used.
Initial value: not specified

-e FILENAME

Sends error messages to FILENAME.
Initial value: not specified
If FILENAME is not specified, the errors are not output.

-f FILENAME

Defines which configuration file EASEL OS/2 EE is to use.
Initial value: not specified. By default, the filename is CONFIG.ESL, and it is searched for in the directories in the OS/2 PATH specification.

-g SIZE

Specifies size, in bytes, of allocated global memory.
Initial value: not specified. This overrides the value of GLBSIZE specified in the configuration file CONFIG.ESL.

-m SIZE

Specifies size, in bytes, of allocated EASEL OS/2 EE program memory.
Initial value: not specified. This overrides the value of PRGSIZE specified in the configuration file CONFIG.ESL.

-nb

When specified, turns off pointing device sounds ("beep" for correct touches and "boop" for mistouches).
Initial value: not specified. You can change the frequency of these sounds in the CONFIG.ESL file.

-terp

Causes EASEL OS/2 EE to generate trace messages for the entire EASEL OS/2 EE program or until a **turn trace off** action statement is encountered. These trace messages are useful for following EASEL OS/2 EE program execution at a low level. When using this option, the **-e** option should also be used. (The **-terp** option works in exactly the same way as the **turn trace on** action statement, except that **-terp** traces the entire program from the beginning, only stopping if a **turn trace off** statement is encountered.)
Initial value: not specified

Chapter 24. Debugging

24.1 Overview

This chapter presents the syntax of EASEL OS/2 EE action statements and command line options that help you locate errors in your EASEL OS/2 EE programs. They are:

turn trace on/off	Action Statement
send to errorlog	Action Statement
-apio	Command Line Option
-terp	Command Line Option

24.1.1 Using the Errorlog File

The errorlog is a file in which EASEL OS/2 EE reports error conditions. Each time an EASEL OS/2 EE program is compiled or executed, an errorlog is produced. If any error message is encountered during compilation, the .EBI file will not be produced; you must fix the errors and recompile. If there are no errors or only warning messages, then the .EBI file is produced. If the **-e** command line option is used and "errors" is specified as the filename, you can examine the entire contents of the errorlog by entering the following command at the OS/2 command prompt:

```
type errors
```

24.1.2 Reading Errors on Another Terminal

You can use an ASCII terminal attached to a communications port to output the contents of the errorlog file. To do this, follow these steps:

1. Connect an ASCII terminal to a communications port. Set the terminal communication speed, and set no parity.
2. Execute a MODE command to initialize the communications port:

```
mode com1:4800,n,8,1
```

3. Run EASEL OS/2 EE, including the following errorlog statement on the command line:

```
easel PROGRAM_NAME -e comN
```

where PROGRAM_NAME is the name of the EASEL OS/2 EE program, and N is the number of the communications port. For example:

```
easel testprog -e com1
```

24.2 Action Statements and Environment Declarations for Debugging

24.2.1 Tracing Action Statements for Portions of an EASEL OS/2 EE Program

You can use the **turn trace on** statement to tell EASEL OS/2 EE to record, in the errorlog, every action statement that it performs. Typically, you trace only those areas of the EASEL OS/2 EE program that are untested or that are not producing the results you want, because EASEL OS/2 EE operates much more slowly when tracing is on. (See also the **-terp** command line option, described in 24.3.2, "Tracing Action Statements for an Entire EASEL OS/2 EE Program" on page 24-5.)

Model

```
[turn] trace { on | off }
```

EASEL OS/2 EE begins to record its actions as soon as it encounters a **turn trace on** statement. For example:

```
response to NewShade
  turn trace on
  copy "red" to Color
  make Box1 color Color
  turn trace off
```

produces errorlog text such as the following:

```
Turning trace on
Copying 'red' to Color
Making Box1 red
Turning trace off
```

To stop the trace, use a **turn trace off** statement. EASEL OS/2 EE will cease tracing as soon as the statement is executed.

24.2.2 Sending Values to the Errorlog

You can send messages to the errorlog from within the EASEL OS/2 EE program, using the following model.

send VALUE1 [VALUE2]... **to errorlog**

Text is written to the errorlog exactly as it is specified, so you may wish to use control characters to control where the text is printed. For example, if you want to send the values of the variables Var1 and Var2 to the errorlog, you might specify the following statement:

```
send "Var1 is: " Var1 "\nVar2 is: " Var2 "\n"
to errorlog
```

Note that the use of "\n" places output on separate lines. This statement produces messages such as the following:

```
Var1 is: Goodbye
Var2 is: Hello
```

You can send any number of values in a single **send** statement.

24.3 Command Line Options for Debugging

24.3.1 Tracing Applications Communications

The **-apio** command line option (short for “**a**pplication **i**n/**o**ut”) causes EASEL OS/2 EE to trace everything that is sent to and received from each application, and record it in the errorlog. You specify the **-apio** command in the command line when you execute an EASEL OS/2 EE program, or add it to the command line in the EASEL.CMD file. For example, if **-apio** is specified, the following statements:

```
response to line "ready" from Recorder
    send "3\n" to Recorder
```

produce errorlog text such as the following:

```
Recorder says: ready ^M^J
Sending '3\n' to Recorder
```

24.3.2 Tracing Action Statements for an Entire EASEL OS/2 EE Program

The **-terp** command line option causes EASEL OS/2 EE to trace every action statement and record it in the errorlog. You specify the **-terp** command in the command line when you execute an EASEL OS/2 EE program, or add it to the command line in the EASEL.CMD file.

The **-terp** option works in exactly the same way as the **turn trace on** action statement, except that **-terp** traces the entire program from the beginning, only stopping if a **turn trace off** statement is encountered.

Chapter 25. Writing Local Applications and Libraries

25.1 Writing Well-behaved Local Applications

The following requirements apply to writing a well-behaved EASEL OS/2 EE local application.

25.1.1 Standalone Program

The local application must be able to be run as a standalone program from the OS/2 fullscreen or windowed command prompt. It cannot be a PM application or a “real-mode” application (that is, a program that runs under DOS or the OS/2 Compatibility Box). The local application should be completely debugged as a standalone program before it is tested with EASEL OS/2 EE.

25.1.2 Keyboard and Screen I/O

I/O to the keyboard and screen must be performed only via OS/2 File-System functions (such as “DosRead” and “DosWrite”) or via high-level language I/O services that translate to OS/2 File-System function calls, such as the “C” Standard I/O library calls. I/O to the keyboard must not be performed via the OS/2 Keyboard functions (those prefixed with “Kbd”). I/O to the screen must not be performed via the OS/2 Video I/O functions (those prefixed with “Vio”).

EASEL OS/2 EE communicates with local applications by intercepting the application’s I/O requests to standard input (the keyboard) and standard output (the screen).

25.1.3 Guidelines for Writing C Local Applications for EASEL OS/2 EE

25.1.3.1 setbuf Function Calls

Always call the following functions at the beginning of function “main()”:

```
setbuf(stdout, NULL);  
setbuf(stderr, NULL);
```

CAUTION:

If you do not make these function calls, your local application will not work properly.

These calls will cause I/O to the “standard output” and “standard error” file streams to be unbuffered. You must also include “stdio.h” at the beginning of your main module.

Note: Any characters written to “stderr” will appear in the EASEL OS/2 EE errorlog file.

25.2 Writing Library Functions and Subroutines

EASEL OS/2 EE supports the ability to call C routines from EASEL OS/2 EE programs. The modules are compiled and linked into an OS/2 Dynamic Link Library whose routines can then be directly referenced in an EASEL OS/2 EE program after they have been declared. These C routines can have integer, floating point, and string arguments, and can return integer, floating point, boolean, or string values. EASEL OS/2 EE distinguishes between C routines that pass a single result back to their caller through their return value and those that operate on their parameters. The former are referred to as functions, the latter as subroutines.

25.2.1 Declaring C Routines In an EASEL OS/2 EE Program

In order to call a C routine that is in a dynamic link library from an EASEL OS/2 EE program, it must first be declared (just as variables must be declared before use). The function declaration model is as follows:

Model (For Functions)

```
function FUNCT_NAME( [TYPE1:ARG_NAME[,  
                      TYPE2:ARG2_NAME ]... ] )  
  
    returns TYPE  
    library LIBRARY
```

The subroutine declaration model is as follows:

Model (For Subroutines)

```
subroutine SUB_NAME( [TYPE1:ARG1_NAME[,  
                     TYPE2:ARG2_NAME]... ] )  
  
    library LIB_NAME
```

Where: FUNCT_NAME and SUB_NAME are C routine names, and must begin with a capital letter.

TYPE_n is: **integer**, **string**, **boolean**, or **float**.

ARG_n_NAME is a variable of TYPE_n.

TYPE is the return type of the function (integer, string, boolean, or float).

LIB_NAME is the name of a dynamic link library. The name should not include the ".dll" extension.

There can be zero or more arguments specified, separated by commas. The ARGn is just a descriptive name indicating the intended use of that parameter, and need not be a variable in the EASEL OS/2 EE program.

25.2.2 Function Arguments and Return Values

Functions take their arguments by value and return a single value as a result. For example, the SQRT() function takes as its argument a float value and returns the value's square root, also as a float value. The following table shows the correspondence between the argument or return type declared in the EASEL OS/2 EE program and the C data type passed to or returned by the function:

EASEL OS/2 EE Type	C Type
Integer	Long
Boolean (return value)	Long
Float	Double
String	HSTRING (see 25.2.4, "Strings" on page 25-5)

A function is called from an EASEL OS/2 EE program using the following model.

Model

```
FUNC_NAME( [ARG1[, ARG2]... ] )
```

Where: ARG1[, ARG2]... are EASEL OS/2 EE expressions.

A function call can be placed in any location where a variable of the same type could be used. For example:

copy (SQRT(15) + 1) to X

25.2.3 Subroutine Arguments and Return Values

Subroutine arguments are passed by reference. Each argument passed to the subroutine is a pointer that corresponds to a variable used as a parameter to the subroutine call. The pointer points to the data stored in the variable. The following table shows the correspondence between the EASEL OS/2 EE variable type and the C data type passed to the subroutine:

EASEL OS/2 EE Type	C Type
Integer	Long*
Float	Double*
String	PHSTRING (See 25.2.4, "Strings.")

For integer and float arguments, the subroutine can return a value by assigning a value indirectly through the pointer argument. Assignment to string variables must be performed through the string-handling API discussed in 25.2.4, "Strings": direct assignment will cause errors.

All subroutines must return a C integer value. The return value is available to the EASEL OS/2 EE program through the built-in function **errorlevel**. This returned quantity should be zero if the C subroutine encountered no errors, or nonzero if an error was generated. The **errorlevel** function can be interrogated by the calling program after it is called, to determine if an error was generated during the subroutine call.

25.2.4 Strings

Several routines included with EASEL OS/2 EE in a dynamic link library allow manipulation of EASEL OS/2 EE string variables in C dynamic link library routines:

EslCreateString(string_length, characters)

```
USHORT  string_length; /* length of string (excluding
                        null terminator) */
PCHAR   characters;    /* pointer to buffer containing
                        initial contents */
```

Description: Allocate and initialize an EASEL OS/2 EE string variable.

Returns: Pointer to an EASEL OS/2 EE string variable allocated and initialized (HSTRING); NULL if the allocation fails.

EslQueryStringAddr(esl_string)

```
HSTRING esl_string; /* an EASEL OS/2 EE string */
```

Description: Translate an EASEL OS/2 EE string variable into a pointer to a (NULL terminated) string of characters.

Returns: Pointer to a NULL terminated string of chars (PCHAR).

EslQueryStringChars(esl_string, buffer, buffer_size)

```
HSTRING esl_string; /* EASEL OS/2 EE string
                    variable */
PCHAR   buffer;      /* pointer to buffer to receive
                    characters */
USHORT  buffer_size; /* size of target buffer */
```

Description: Copy characters from an EASEL OS/2 EE string variable into the specified buffer.

Returns: Count of characters copied (USHORT).

`EslSetStringValue(esl_string, buffer, count)`

```
PHSTRING esl_string; /* pointer to an EASEL OS/2 EE
                      string variable */
PCHAR    buffer;     /* pointer to buffer containing
                      initial contents */
USHORT    count;     /* count of characters to
                      copy */
```

Description: Copy count characters from the specified buffer into an EASEL OS/2 EE string variable.

Returns: Pointer to a (NULL terminated) string of characters (PCHAR); NULL if the allocation fails.

`EslSetStringHandle(esl_string1, esl_string2)`

```
PHSTRING target_esl_string; /* pointer to an EASEL
                             OS/2 EE string
                             variable */
HSTRING  source_esl_string; /* EASEL OS/2 EE string
                             variable */
```

Description: Set an EASEL OS/2 EE string variable `target_esl_string` to EASEL OS/2 EE string variable `source_esl_string`.

Returns: TRUE (USHORT).

Usage of EASEL OS/2 EE library routines:

- Use `EslCreateString` when a function is to return an EASEL OS/2 EE string variable.
- Use `EslQueryStringAddr` to get to the contents of an EASEL OS/2 EE string variable directly.
- Use `EslQueryStringChars` to copy the contents of an EASEL OS/2 EE string variable to a local buffer.
- Use `EslSetStringValue` to copy characters into an EASEL OS/2 EE string variable.
- Use `EslSetStringHandle` to copy an EASEL OS/2 EE string variable into another EASEL OS/2 EE string variable directly.

25.2.5 C Coding Considerations

All library modules should include the file OS2.H. If the module uses EASEL OS/2 EE string values, include the file ESLLIB.H.

Create a module definitions file to export your function. It should contain the following:

```
LIBRARY      eslaux
DESCRIPTION  'EASEL OS/2 EE Auxiliary Routines'

DATA  PRELOAD
CODE  PRELOAD

HEAPSIZE 0

EXPORTS
    Its_EASEL_Name1 = its_c_name1 @1
    Its_EASEL_Name2 = its_c_name2 @2
```

In this example, the module definitions file is called "eslaux.def" and the library is called "eslaux.dll". The EXPORTS section defines the C routines for the library. The first entry is the name of the routine as it is known to EASEL OS/2 EE, followed by an equals sign, followed by the routine's C name. The last field on each export line must be a unique number preceded by the @ symbol. These numbers should be in order, without omission; that is, the first entry should be numbered 1; the second 2; and so on.

The C routines must be declared as APIENTRY. For example: if a function called Bigger returns 1 or 0, it should be declared as follows:

```
USHORT APIENTRY Bigger( int1, int2 )
{
    long int1;
    long int2;

    if ( int1 > int2 )
        return( 1 );
    else
        return( 0 );
}
```

The flags used to compile are specific:

-Zpe1 -G2s -Aflu

The link line used is also specific:

link /A:16/NOD bigger,biglib,biglib,os2+llibcdll,biglib

where the module is named bigger.c and the library will be called biglib.dll. If you want to use any EASEL OS/2 EE string variable manipulation dynamic link library routines, the link line would be:

```
link /A:16/NOD bigger, biglib, biglib,  
      os2+llibcdll+esllib,biglib
```

Certain common C runtime library routines cannot be used in dynamic link library routines. Only C runtime routines that assume that ds and ss are not equal can be used in dynamic link libraries. Among those that assume that ds and ss are equal are _exit, exit, abort, chsize, creat, tmpfile, the exec functions, the printf and scanf functions, and the spawn functions.

You can usually find a workaround; for example, using write() instead of fprintf(). If you have a problem using a C runtime library routine, test it by creating a program that contains a main() routine that calls the problem C runtime library routine. If the new program works as expected and the dynamic link library does not, the C runtime library routine is probably not callable from within a dynamic link library.

25.2.6 Example Library and Usage

25.2.6.1 Example C Module

The following is a comprehensive example of a user-written dynamic link library consisting of two subroutines and two functions.

Note the use of the return value for the subroutines.

```
#include <os2def.h>  
#include <esllib.h>  
#include <ctype.h>  
#include <stdlib.h>  
  
/*  
 * Take the next line out of comments to print out  
 * the debugging messages.  
 */  
  
/*#define DEBUG*/  
  
#ifdef DEBUG  
extern char *itoa();  
      char tmp[33];  
  
#define WRITE_IT_OUT(a)   write(2, (a), strlen((a)))  
#define NEWLINE           write(2, "\n," 1)  
  
char str_buff[33]; /* global string buffer */  
#endif
```

```

/* * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/*  return TRUE if there is an uppercase alphabetic  */
/*    character in the EASEL OS/2 EE string passed  */
/*                                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * */
LONG APIENTRY
look_for_upper(esl_str)
HSTRING esl_str;
{
    int i;
    char *str;

    /*
     * get pointer to string passed
     */
    str = EslQueryStringAddr(esl_str);

    /*
     * examine each character until all
     * examined or uppercase alpha found
     */
    for (i = 0; i < strlen(str); i++) {
        if ( isupper((int)*str++) )
            return ( TRUE );
    }
    return ( FALSE );
}

/* * * * * * * * * * * * * * * * * * * * * * * */
/*                                                                 */
/*  return TRUE if there is a lowercase alphabetic  */
/*    character in the EASEL OS/2 EE string passed  */
/*                                                                 */
/* * * * * * * * * * * * * * * * * * * * * * * */
LONG APIENTRY
look_for_lower(esl_str)
HSTRING esl_str;
{
    int i;
    char *str;

    /*
     * get pointer to string passed
     */

```

```

    str = EslQueryStringAddr(esl_str);

    /*
     * examine each character until all
     * examined or lowercase alpha found
     */
    for (i = 0; i < strlen(str); i++ ) {
        if ( islower((int)*str++) )
            return ( TRUE );
    }
    return ( FALSE );
}

/* * * * * * * * * * * * * * * * * * * * * * * */
/*
/* Convert all lowercase alphabetic characters in the */
/* EASEL OS/2 EE string passed to uppercase characters */
/*
/* * * * * * * * * * * * * * * * * * * * * * */
LONG APIENTRY
convert_to_upper(esl_str)
HSTRING *esl_str;
{
    int i, size, copied;
    char *str, *tmp_str;

    /*
     * get pointer to string passed
     */
    str = EslQueryStringAddr(*esl_str);

    /*
     * get its length
     */
    size = strlen(str);

#ifdef DEBUG
    /* check contents and length of str */
    WRITE_IT_OUT("str = ");
    WRITE_IT_OUT(str);
    WRITE_IT_OUT("size = ");
    itoa(size,tmp,10);
    WRITE_IT_OUT(tmp);
    NEWLINE;
#endif
}

```

```

/*
 * allocate memory for local copy (reuse pointer str)
 */
str = malloc(++size);

/*
 * get copy of string passed
 */
copied = EslQueryStringChars(*esl_str, str, size);

#ifdef DEBUG
/* how many characters were copied */
WRITE_IT_OUT("copied = ");
itoa(copied,tmp,10);
WRITE_IT_OUT(tmp);
NEWLINE;
#endif

/*
 * convert each lowercase alpha
 * found to uppercase alpha
 */
for (i = 0,tmp_str = str; i < size; i++,tmp_str++) {
    *tmp_str = (char)toupper((int)*tmp_str);
}

/*
 * set EASEL OS/2 EE string value
 */
EslSetStringValue(esl_str, str, --size);

/*
 * release local memory
 */
free(str);

return((LONG)i);
}

/* * * * * * */
/*
/* Convert all uppercase alphabetic characters in the */
/* EASEL OS/2 EE string passed to lowercase characters */
/*
/* * * * * * */

```

```

LONG APIENTRY
convert_to_lower(esl_str)
HSTRING *esl_str;
{
    int i, size, copied;
    char *str, *tmp_str;

    /*
     * get pointer to string passed
     */
    str = EslQueryStringAddr(*esl_str);

    /*
     * get its length
     */
    size = strlen(str);

#ifdef DEBUG
    /* check contents and length of str */
    WRITE_IT_OUT("str = ");
    WRITE_IT_OUT(str);

    WRITE_IT_OUT("size = ");
    itoa(size,tmp,10);
    WRITE_IT_OUT(tmp);
    NEWLINE;
#endif

    /*
     * allocate memory for local copy (reuse pointer
str)
     */
    str = malloc(++size);

    /*
     * get copy of string passed
     */
    copied = EslQueryStringChars(*esl_str, str, size);

#ifdef DEBUG
    WRITE_IT_OUT("copied = ");
    itoa(copied,tmp,10);
    WRITE_IT_OUT(tmp);
    NEWLINE;
#endif

    /*

```



```

    * convert each uppercase alpha
    * found to lowercase alpha
    */
    for (i = 0,tmp_str = str; i < size; i++,tmp_str++) {
        *tmp_str = (char)tolower((int)*tmp_str);
    }

    /*
    * set EASEL OS/2 EE string value
    */
    Es1SetStringValue(esl_str, str, --size);

    /*
    * release local memory
    */

    free(str);

    return((LONG)i);
}

```

25.2.6.2 Makefile

The -I flag is included because c:\EASEL-EE is where ESLLIB.H is stored in the environment in which this makefile is run. If you store your ESLLIB.H somewhere else, simply change the -I flag to that subdirectory. The esllib.lib file is on c:\EASEL-EE, so in the link line the full pathname is specified.

```
CFLAGS = -c -Ic:\easel-ee -W2 -J -Zpe1 -G2s -Aflu
```

```
OBJECTS = example.obj
```

```
SOURCE = example.c
```

```
example.exe: $(SOURCE) $(OBJECTS) example.def
    link /a:16 /nod example, example.dll,-
    os2+llibcdll+c:\easel-ee\esllib, example.def

```

If your library consists of more than one module, add the additional entries to the OBJECTS entry and the SOURCE entry, and in the link line. For example:

```
CFLAGS = -c -Ic:\easel-ee -W2 -J -Zpe1 -G2s -Aflu
```

```
OBJECTS = example.obj example2.obj
```

```
SOURCE = example.c example2.c
```

```
example.exe: $(SOURCE) $(OBJECTS) example.def  
    link /a:16 /nod example+example2, example.dll,,  
    os2+1libcdll+c:\easel-ee\esllib,example.def
```

25.2.6.3 Definition File

```
LIBRARY      example  
DESCRIPTION  'EASEL OS/2 EE Example Routines'
```

```
DATA  PRELOAD
```

```
CODE  PRELOAD
```

```
EXPORTS
```

```
    HasUpper = look_for_upper    @1  
    MakeUpper = convert_to_upper @2  
    HasLower = look_for_lower    @3  
    MakeLower = convert_to_lower @4
```

25.2.6.4 EASEL OS/2 EE Program

Note the ability to check the return value of the subroutines using the EASEL OS/2 EE built-in function **errorlevel**.

```
function HasUpper(string:MixedCaseString) returns boolean  
    library "example"  
function HasLower(string:MixedCaseString) returns boolean  
    library "example"  
subroutine MakeUpper(string:MixedCaseString)  
    library "example"  
subroutine MakeLower(string:MixedCaseString)  
    library "example"
```

```
string AllCaps is "THIS STRING IS ALL CAPS"  
string AllLower is "this string is all lower"  
integer ErrorLevel  
string Mixed is "This STRiNG IS all miXed Up"  
string GuineaPig
```

```
response to start  
    send "HasUpper(AllCaps) = " to errorlog
```

```

if ( HasUpper( AllCaps ) ) then
    send "TRUE\n" to errorlog
else
    send "FALSE\n" to errorlog
end if

send "HasUpper(AllLower) = " to errorlog
if ( HasUpper( AllLower ) ) then
    send "TRUE\n" to errorlog
else
    send "FALSE\n" to errorlog
end if

send "HasUpper(Mixed) = " to errorlog
if ( HasUpper( Mixed ) ) then
    send "TRUE\n" to errorlog
else
    send "FALSE\n" to errorlog
end if

send "HasLower(AllCaps) = " to errorlog
if ( HasLower( AllCaps ) ) then
    send "TRUE\n" to errorlog
else
    send "FALSE\n" to errorlog
end if

send "HasLower(AllLower) = " to errorlog
if ( HasLower( AllLower ) ) then
    send "TRUE\n" to errorlog
else
    send "FALSE\n" to errorlog
end if

send "HasLower(Mixed) = " to errorlog
if ( HasLower( Mixed ) ) then
    send "TRUE\n" to errorlog
else
    send "FALSE\n" to errorlog
end if

copy "THIS STRING IS ALL CAPS" to GuineaPig
send "GuineaPig BEFORE MakeUpper: " GuineaPig "\n"
    to errorlog
call MakeUpper(GuineaPig)
copy errorlevel to ErrorLevel
send "GuineaPig AFTER MakeUpper: " GuineaPig "\n"
    to errorlog
send "ErrorLevel is: " ErrorLevel "\n\n" to errorlog

```

```

copy "this string is all lower" to GuineaPig
send "GuineaPig BEFORE MakeUpper: " GuineaPig "\n"
  to errorlog
call MakeUpper(GuineaPig)
copy errorlevel to ErrorLevel
send "GuineaPig AFTER MakeUpper: " GuineaPig "\n"
  to errorlog
send "ErrorLevel is: " ErrorLevel "\n\n" to
  errorlog

```

```

copy "This STRiNG IS aLL miXed Up" to GuineaPig
send "GuineaPig BEFORE MakeUpper: " GuineaPig "\n"
  to errorlog
call MakeUpper(GuineaPig)
copy errorlevel to ErrorLevel
send "GuineaPig AFTER MakeUpper: " GuineaPig "\n"
  to errorlog
send "ErrorLevel is: " ErrorLevel "\n\n" to errorlog

```

```

copy "THIS STRING IS ALL CAPS" to GuineaPig
send "GuineaPig BEFORE MakeLower: " GuineaPig "\n"
  to errorlog
call MakeLower(GuineaPig)
copy errorlevel to ErrorLevel
send "GuineaPig AFTER MakeLower: " GuineaPig "\n"
  to errorlog
send "ErrorLevel is: " ErrorLevel "\n\n" to errorlog

```

```

copy "and this string is all lower, too" to GuineaPig
send "GuineaPig BEFORE MakeLower: " GuineaPig "\n"
  to errorlog
call MakeLower(GuineaPig)
copy errorlevel to ErrorLevel
send "GuineaPig AFTER MakeLower: " GuineaPig "\n"
  to errorlog
send "ErrorLevel is: " ErrorLevel "\n\n" to errorlog

```

```

copy "But This STRiNG IS aLL miXed Up compLETly"
  to GuineaPig
send "GuineaPig BEFORE MakeLower: " GuineaPig "\n"
  to errorlog
call MakeLower(GuineaPig)
copy errorlevel to ErrorLevel
send "GuineaPig AFTER MakeLower: " GuineaPig "\n"
  to errorlog
send "ErrorLevel is: " ErrorLevel "\n\n" to errorlog
exit

```

25.2.6.5 Output

The following output will appear in the errorlog file (if one has been specified with the `-e` command line option).

```
HasUpper(AllCaps) = TRUE
HasUpper(AllLower) = FALSE
HasUpper(Mixed) = TRUE
HasLower(AllCaps) = FALSE
HasLower(AllLower) = TRUE
HasLower(Mixed) = TRUE
GuineaPig BEFORE MakeUpper: THIS STRING IS ALL CAPS
GuineaPig AFTER MakeUpper: THIS STRING IS ALL CAPS
ErrorLevel is: 24
```

```
GuineaPig BEFORE MakeUpper: this string is all lower
GuineaPig AFTER MakeUpper: THIS STRING IS ALL LOWER
ErrorLevel is: 25
```

```
GuineaPig BEFORE MakeUpper: This STRiNG IS all miXed Up
GuineaPig AFTER MakeUpper: THIS STRING IS ALL MIXED UP
ErrorLevel is: 28
```

```
GuineaPig BEFORE MakeLower: THIS STRING IS ALL CAPS
GuineaPig AFTER MakeLower: this string is all caps
ErrorLevel is: 24
```

```
GuineaPig BEFORE MakeLower: and this string is all lower, too
GuineaPig AFTER MakeLower: and this string is all lower, too
ErrorLevel is: 34
```

```
GuineaPig BEFORE MakeLower: But This STRiNG
IS all miXed Up comPLETly
GuineaPig AFTER MakeLower: but this string is all
mixed up completly
ErrorLevel is: 42
```

25.3 Writing a Stimulus Library

A stimulus library must conform to the following specifications, in addition to those required for a normal EASEL OS/2 EE DLL (see 25.2, "Writing Library Functions and Subroutines" on page 25-3).

- The DLL must export two entry points as ordinals 1 and 2, which will be used, respectively, for initialization and termination. Each entry point must be an APIENTRY (i.e., "far pascal") function and must return an integer 0.
- The initialization routine is called by EASEL OS/2 EE immediately after loading the first application that references the DLL as a stimulus. The initialization routine must take a single argument, which is a PM window handle (HWND). This window handle is used for passing stimulus events to the EASEL OS/2 EE program.
- The termination routine must take the same PM window handle as an argument. The termination routine is called just before exiting from the EASEL OS/2 EE runtime, giving the DLL a chance to perform any necessary cleanup operations.
- The DLL may have additional entry points to be called explicitly from the EASEL OS/2 EE program.
- Whenever the DLL wishes to pass a stimulus event to the EASEL OS/2 EE program, it must post a message (using WinPostMsg) to the window handle received in the initialization function. The message must not be sent (using WinSendMsg), and should not be posted after the termination routine has been executed.
- The message posted must be an unsigned short integer whose value is (WM_USER + **eventnumber**), where **eventnumber** is the value that EASEL OS/2 EE uses for performing event matching and that is returned in the EASEL OS/2 EE built-in function **eventnumber**. Message parameter one (mp1) must be a long integer; this value is returned in the built-in function **eventparam** and is used for event parameter matching. Message parameter two (mp2) is reserved and must be NULL.

Appendix A. Include Files

A.1 Overview

This appendix contains the include files that are supplied with EASEL OS/2 EE.

A.2 accel.inc file

```
#####  
#                                                                 #  
# Name: ACCEL.INC                                              #  
#                                                                 #  
# Constants for defining accelerator keys.                    #  
#                                                                 #  
# (C) Copyright Interactive Images, Inc. 1989, 1990          #  
# All Rights Reserved                                         #  
#                                                                 #  
#####  
  
integer constant KEY_BREAK          is 261  
integer constant KEY_BACKSPACE     is 262  
integer constant KEY_TAB           is 263  
integer constant KEY_BACKTAB       is 264  
integer constant KEY_PAUSE         is 270  
integer constant KEY_CAPSLOCK      is 271  
integer constant KEY_ESC           is 272  
integer constant KEY_SPACE         is 273  
integer constant KEY_PAGEUP        is 274  
integer constant KEY_PAGEDOWN      is 275  
integer constant KEY_END           is 276  
integer constant KEY_HOME          is 277  
integer constant KEY_LEFT          is 278  
integer constant KEY_UP            is 279  
integer constant KEY_RIGHT         is 280  
integer constant KEY_DOWN          is 281  
integer constant KEY_PRINTSCRN     is 282  
integer constant KEY_INSERT        is 283  
integer constant KEY_DELETE        is 284  
integer constant KEY_SCROLLLOCK    is 285  
integer constant KEY_NUMLOCK       is 286  
integer constant KEY_ENTER         is 287  
integer constant KEY_SYSRQ         is 288  
integer constant KEY_F1            is 289  
integer constant KEY_F2            is 290  
integer constant KEY_F3            is 291  
integer constant KEY_F4            is 292  
integer constant KEY_F5            is 293  
integer constant KEY_F6            is 294  
integer constant KEY_F7            is 295  
integer constant KEY_F8            is 296  
integer constant KEY_F9            is 297  
integer constant KEY_F10           is 298  
integer constant KEY_F11           is 299  
integer constant KEY_F12           is 300
```

A.3 datelib.inc file

```
#####  
#  
# Name: DATELIB.INC  
#  
# Declarations for day/date library routines.  
#  
# (C) Copyright Interactive Images, Inc. 1989, 1990  
# All Rights Reserved  
#  
#####  
  
integer constant PMDate is      1  
integer constant OS2Date is    2  
integer constant PMTime is     1  
integer constant OS2Time is    2  
  
function Validdate(integer:Year, integer:Month, integer:Day)  
                                returns boolean library "datelib"  
function WeekDay(integer:Year, integer:Month, integer:Day)  
                                returns string library "datelib"  
function DaySpan(integer:Year, integer:Month, integer:Day, integer:Year,  
                  integer:Month, integer:Day)  
                                returns integer library "datelib"  
function LocalDate(integer:Type)  
                                returns string library "datelib"  
function LocalTime(integer:Type)  
                                returns string library "datelib"
```

A.4 egacolor.inc File

```
#####  
#                                                                 #  
# Name: EGACOLOR.INC                                           #  
#                                                                 #  
# Declarations for full-screen EGA/VGA color palette functions. #  
#                                                                 #  
# (C) Copyright Interactive Images, Inc. 1989, 1990           #  
# All Rights Reserved                                           #  
#                                                                 #  
#####  
  
# this file is now obsolete, and is maintained for backward  
# compatibility, the contents of this file are now in esllib.inc  
  
include "esllib.inc"
```

A.5 eslappc.inc file

```
#####  
#  
# Name: ESLAPPC.INC  
#  
# Declarations for APPC subroutines and constants  
#  
# (C) Copyright International Business Machines Corporation 1990.  
# All Rights Reserved  
#  
#####  
  
# define EASEL OS/2 EE APPC Subroutines callable from EASEL OS/2 EE  
  
subroutine APPCAcceptStart(integer:Conversation_ID,  
                           string:Profile_Name,  
                           integer:State_Switch)  
    library "eslappc"  
  
subroutine APPCConvertA2E(integer:Conversation_ID,  
                           string:String,  
                           integer:Index_Start,  
                           integer:Index_End,  
                           integer:ASCII_Table,  
                           integer:EBCDIC_Table)  
    library "eslappc"  
  
subroutine APPCConvertE2A(integer:Conversation_ID,  
                           string:String,  
                           integer:Index_Start,  
                           integer:Index_End,  
                           integer:ASCII_Table,  
                           integer:EBCDIC_Table)  
    library "eslappc"  
  
subroutine APPCEnd(integer:Conversation_ID,  
                   integer:Sync_Level)  
    library "eslappc"  
  
subroutine APPCFlush(integer:Conversation_ID)  
    library "eslappc"  
  
subroutine APPCGetAcceptProfile(string:Profile_Name,  
                                string:Invocation_CMD,  
                                integer:Inactivity_Timeout,  
                                integer:Buffer_Size,  
                                integer:ASCII_Table,  
                                integer:EBCDIC_Table)  
    library "eslappc"  
  
subroutine APPCGetAcceptProfileNames(string:Profile_Names)  
    library "eslappc"
```

```

subroutine APPCGetInfo(integer:Conversation_ID,
                      string:APPC_TP_ID,
                      integer:APPC_Conversation_ID,
                      string:NET_Name,
                      string:Local_LU_Name,
                      string:Local_LU_Alias_Name,
                      string:Partner_LU_Alias_Name,
                      string:Partner_LU_Uninterpreted_Name,
                      string:FullyQual_Partner_LU_Name,
                      string:Partner_TP_Name,
                      string:Mode_Name,
                      integer:Sync_Level,
                      string:User_ID)
    library "eslappc"

subroutine APPCGetInitProfile(string:Profile_Name,
                              string:Local_LU_Alias_Name,
                              string:Partner_LU_Alias_Name,
                              string:Partner_TP_Name,
                              string:Mode_Name,
                              string:Security_Parameters,
                              integer:Inactivity_Timeout,
                              integer:Buffer_Size,
                              integer:ASCII_Table,
                              integer:EBCDIC_Table)
    library "eslappc"

subroutine APPCGetInitProfileNames(string:Profile_Names)
    library "eslappc"

subroutine APPCGetString(integer:Conversation_ID,
                          string:String_Var,
                          integer:Auto_Convert)
    library "eslappc"

subroutine APPCGetSystemError(integer:Conversation_ID,
                              integer:Work_Request_Error,
                              integer:Work_Request_Identifier,
                              integer:APPC_Verb,
                              integer:Primary_Error_Code,
                              integer:Secondary_Error_Code)
    library "eslappc"

subroutine APPCReadyToReceive(integer:Conversation_ID)
    library "eslappc"

subroutine APPCReceiveNotify(integer:Conversation_ID)
    library "eslappc"

subroutine APPCRequestConfirm(integer:Conversation_ID)
    library "eslappc"

subroutine APPCRequestToSend(integer:Conversation_ID)
    library "eslappc"

```

```

subroutine APPCSendConfirmed(integer:Conversation_ID)
    library "eslappc"

subroutine APPCSendError(integer:Conversation_ID,
                        integer>Error_Origin)
    library "eslappc"

subroutine APPCSendString(integer:Conversation_ID,
                        string:String_Var,
                        integer:Auto_Convert,
                        integer:Send_Ind,
                        integer:More_to_Send)
    library "eslappc"

subroutine APPCSetAcceptProfile(string:Profile_Name,
                        integer:Action,
                        string:Invocation_CMD,
                        integer:Inactivity_Timeout,
                        integer:Buffer_Size,
                        integer:ASCII_Table,
                        integer:EBCDIC_Table)
    library "eslappc"

subroutine APPCSetInitProfile(string:Profile_Name,
                        integer:Action,
                        string:Local_LU_Alias_Name,
                        string:Partner_LU_Alias_Name,
                        string:Partner_TP_Name,
                        string:Mode_Name,
                        string:Security_Parameters,
                        integer:Inactivity_Timeout,
                        integer:Buffer_Size,
                        integer:ASCII_Table,
                        integer:EBCDIC_Table)
    library "eslappc"

subroutine APPCStart(integer:Conversation_ID,
                    string:Profile_Name,
                    integer:Sync_Level,
                    integer:State_Switch)
    library "eslappc"

subroutine APPCTestReqToSend(integer:Conversation_ID)
    library "eslappc"

# define EASEL OS/2 EE APPC Messages sent to EASEL OS/2 EE

integer constant APPC_Confirmed      is 5
integer constant APPC_ConfirmRequest is 4
integer constant APPC_Data           is 3
integer constant APPC_Ended          is 2
integer constant APPC_EndReceive     is 12
integer constant APPC_InputReceived  is 11
integer constant APPC_SendError      is 9
integer constant APPC_ReceiveError   is 8
integer constant APPC_RequestToSend is 6
integer constant APPC_SendState      is 7

```

```
integer constant APPC_Started      is 1
integer constant APPC_SystemError  is 10
```

define EASEL OS/2 EE APPC Constants used in Subroutine Calls

```
integer constant APPC_Default      is 0
integer constant APPC_None         is 1
integer constant APPC_Confirm      is 2
integer constant APPC_Automatic    is 3
integer constant APPC_Controlled   is 4
integer constant APPC_NoConvert     is 5
integer constant APPC_Convert      is 6
integer constant APPC_Flush        is 7
integer constant APPC_SyncLevel     is 8
integer constant APPC_MoreData     is 9
integer constant APPC_ReadyToReceive is 10
integer constant APPC_Abend        is 11
integer constant APPC_BadGet       is 12
integer constant APPC_BadSend      is 13
integer constant APPC_CreateProfile is 14
integer constant APPC_ReplaceProfile is 15
integer constant APPC_DeleteProfile is 16
```

Return Codes returned by the APPC... Subroutines

```
integer constant APPC_NORMAL_COMPLETION is 0
```

General Errors

```
integer constant APPC_ERR_INFO_NOT_AVAILABLE is 5
integer constant APPC_ERR_STRING_TRUNCATED is 6
integer constant APPC_ERR_ON_FILE_OPEN is 8
integer constant APPC_ERR_ON_FILE_READ is 9
integer constant APPC_ERR_ON_FILE_WRITE is 10
integer constant APPC_ERR_OUT_OF_DISK_SPACE is 12
integer constant APPC_ERR_INV_WINDOW_HDL is 14
integer constant APPC_ERR_BUFFER_TOO_SMALL is 15
integer constant APPC_ERR_OUT_OF_MEMORY is 16
integer constant APPC_ERR_CONVERSION_ERROR is 18
```

Input Parameter Errors

```
integer constant APPC_ERR_INVALID_PARMS is 20
integer constant APPC_ERR_INV_CONV_ID is 21
integer constant APPC_ERR_INV_SYNC_LEVEL is 22
integer constant APPC_ERR_INV_STATE_SWITCH is 23
integer constant APPC_ERR_INV_AUTO_CONVERT is 24
integer constant APPC_ERR_INV_SEND_IND is 25
integer constant APPC_ERR_INV_MORE_TO_SEND is 26
integer constant APPC_ERR_INV_ERROR_ORIGIN is 27
integer constant APPC_ERR_INV_INDEX_START is 28
integer constant APPC_ERR_INV_INDEX_END is 29
integer constant APPC_ERR_INV_INTEGER_VAR is 30
integer constant APPC_ERR_INV_STRING_VAR is 31
integer constant APPC_ERR_INV_PROFILE_NAME is 32
integer constant APPC_ERR_INV_LOC_LU_ALIAS_NAME is 33
integer constant APPC_ERR_INV_PAR_LU_ALIAS_NAME is 34
integer constant APPC_ERR_INV_PAR_TP_NAME is 35
integer constant APPC_ERR_INV_MODE_NAME is 36
integer constant APPC_ERR_INV_SEC_PARMS is 37
integer constant APPC_ERR_INV_INACT_TIMEOUT is 38
integer constant APPC_ERR_INV_BUFFER_SIZE is 39
integer constant APPC_ERR_INV_ASCII_TABLE is 40
integer constant APPC_ERR_INV_EBCDIC_TABLE is 41
integer constant APPC_ERR_INV_PROFILE_ACTION is 42
```

```
integer constant APPC_ERR_INV_INVOC_COMMAND is 43
integer constant APPC_ERR_INV_BUFFER_POINTER is 44
integer constant APPC_ERR_INV_BUFFER_LENGTH is 45
integer constant APPC_ERR_LGTH_EXCDS_PRF_BUFSIZE is 46
```

Request Processing Errors

```
integer constant APPC_ERR_PROFILE_ALREADY_EXISTS is 50
integer constant APPC_ERR_NO_DATA_AVAILABLE is 51
integer constant APPC_ERR_SYSTEM_ERROR_PENDING is 52
integer constant APPC_ERR_INV_TO_ISSUE_CNFRM_REQ is 53
integer constant APPC_ERR_INV_TO_ISSUE_CONFIRMED is 54
integer constant APPC_ERR_CONFIRM_REQ_IGNORED is 55
integer constant APPC_ERR_INV_TO_ISSUE_RECV_NTFY is 56
integer constant APPC_ERR_NO_REMOTE_TP_PARAM is 57
integer constant APPC_ERR_BASIC_CONV_NOT_VALID is 58
integer constant APPC_ERR_NO_SYSTEM_ERROR_DATA is 59
integer constant APPC_ERR_CONV_ENDING_REQ_PURGED is 60
```

Errors when calling EASEL OS/2 EE Subroutines

```
integer constant APPC_ERR_UNEXPECTED_EASEL_ERROR is 70
```

Unexpected miscellaneous errors

```
integer constant APPC_ERR_UNEXPECTED_ERROR is 80
```

Unexpected errors calling Dos... Subroutines

```
integer constant APPC_ERR_UNEXPECTED_DOS_ERROR is 90
```

Errors when calling APPC

```
integer constant APPC_ERROR is 100
```

Values placed in WORK_REQUEST_IDENTIFIER on return from APPCGetSystemError.

```
integer constant APPC_AcceptStart is 35
integer constant APPC_ConvertString is 37
integer constant APPC_GetInfo is 38
integer constant APPC_ReceiveBuffer is 30
integer constant APPC_ReceiveNotify is 25
integer constant APPC_ReceiveString is 28
integer constant APPC_RequestConfirm is 22
integer constant APPC_RequestRdyToReceive is 31
integer constant APPC_RequestSend is 26
integer constant APPC_SendAbend is 36
integer constant APPC_SendConfirmed is 23
integer constant APPC_SendData is 20
integer constant APPC_SendErrorItem is 24
integer constant APPC_SendFlush is 21
integer constant APPC_SendTerminate is 32
integer constant APPC_Start is 34
integer constant APPC_TestReqSend is 27
```

Values placed in APPC_VERB on return from APPCGetSystemError.

```
integer constant AP_M_ALLOCATE is 1
integer constant AP_M_CONFIRM is 3
integer constant AP_M_CONFIRMED is 4
integer constant AP_M_DEALLOCATE is 5
integer constant AP_M_FLUSH is 6
integer constant AP_M_GET_ATTRIBUTES is 7
integer constant AP_M_PREPARE_TO_RECEIVE is 10
integer constant AP_M_RECEIVE_AND_WAIT is 11
integer constant AP_M_RECEIVE_IMMEDIATE is 12
integer constant AP_M_RECEIVE_AND_POST is 13
integer constant AP_M_REQUEST_TO_SEND is 14
integer constant AP_M_SEND_DATA is 15
```


integer constant	AP_M_SEND_ERROR	is	16
integer constant	AP_M_TEST_RTS	is	18
integer constant	AP_RECEIVE_ALLOCATE	is	22
integer constant	AP_TP_ENDED	is	19
integer constant	AP_TP_STARTED	is	20
integer constant	SV_GET_CP_CONVERT_TABLE	is	25

A.6 esaudio.inc file

```
#####  
#  
# Name: ESLAUDIO.INC  
#  
# Declarations for audio functions.  
#  
# (C) Copyright International Business Machines Corporation 1990. #  
# All Rights Reserved  
#  
#####  
  
#  
# Audio Subroutine definitions  
#  
  
subroutine AUDClose      ( ) library "esaudio"  
  
subroutine AUDErase      ( string : AUDIO_FILENAME ) library "esaudio"  
  
subroutine AUDOpen       ( string : AUDIO_PATHNAME ) library "esaudio"  
  
subroutine AUDPause      ( ) library "esaudio"  
  
subroutine AUDPlay       ( string : AUDIO_FILENAME,  
                           integer : PLAY_START_POSITION,  
                           integer : PLAY_STOP_POSITION,  
                           integer : VOLUME,  
                           integer : BALANCE,  
                           integer : BUSY_ACTION ) library "esaudio"  
  
subroutine AUDQueryBusy  ( integer : BUSY_STATUS ) library "esaudio"  
  
subroutine AUDQueryElapsedTime ( integer : ELAPSED_TIME ) library "esaudio"  
  
subroutine AUDQueryPlayingTime ( string : AUDIO_FILENAME,  
                                  integer : PLAYING_TIME ) library "esaudio"  
  
subroutine AUDRecord     ( string : AUDIO_FILENAME,  
                           integer : RECORD_START_POSITION,  
                           integer : RECORD_STOP_POSITION,  
                           integer : SOUND_SOURCE,  
                           integer : SOUND_QUALITY,  
                           integer : BEEP,  
                           integer : BUSY_ACTION ) library "esaudi"  
  
subroutine AUDResume     ( ) library "esaudio"  
  
subroutine AUDSetBalance ( integer : BALANCE ) library "esaudio"  
  
subroutine AUDSetVolume  ( integer : VOLUME ) library "esaudio"  
  
subroutine AUDStop       ( integer : PURGE_ACTION ) library "esaudio"  
  
  
# defaults  
integer constant AUD_DefaultVolume is 100
```

```

integer constant AUD_DefaultBalance is 50
integer constant AUD_DefaultQuality is 1

# BUSY_ACTION values
integer constant AUD_Interrupt is 1 # interrupt current audio
integer constant AUD_Return is 2 # do not interrupt audio
integer constant AUD_Queue is 4 # queue audio request

# SOUND_SOURCE values
integer constant AUD_Mike is 0 # standard mike input
integer constant AUD_LineLeft is 1 # Line left channel only
integer constant AUD_LineRight is 2 # Line right channel only
integer constant AUD_Line is 3 # Line both channels
integer constant AUD_MikeLow is 4 # Low Mike input

# SOUND_QUALITY values
integer constant AUD_MusicQuality is 1 # 11k/sec
integer constant AUD_VoiceQuality is 2 # 5.5k/sec
integer constant AUD_StereoQuality is 3 # 22k/sec

# BEEP values
integer constant AUD_NoBeep is 0
integer constant AUD_Beep is 1

# PURGE_ACTION values
integer constant AUD_NoPurge is 0 # stop and do next audio req
integer constant AUD_Purge is 1 # stop and clear audio queue

# AUDQueryBusy return values
integer constant AUD_NotBusy is 0
integer constant AUD_BusyPlaying is 4
integer constant AUD_BusyRecording is 6
integer constant AUD_BusyPausing is 8

# Audio Stimulus messages (passed in "eventnumber")
integer constant AUD_Start is 3
integer constant AUD_End is 4
integer constant AUD_Error is 7

# AUD_Start and AUD_End parameter filters (passed in "eventparam")
integer constant AUD_Play is 1
integer constant AUD_Record is 2

# AUD_Error parameter filters (passed in "eventparam")
integer constant AUD_OpenError is 8
integer constant AUD_DiskFull is 22

# return codes
integer constant AUD_NORMAL_COMPLETION is 0
integer constant AUD_ERR_BUSY is 4
integer constant AUD_ERR_NOT_BUSY is 4
integer constant AUD_ERR_ALREADY_DONE is 4
integer constant AUD_ERR_OPEN is 8
integer constant AUD_ERR_QUEUE_FULL is 16
integer constant AUD_ERR_INVALID_PARMS is 20

```

A.7 esllib.inc file

```
#####  
#  
# Name: ESLLIB.INC  
#  
# Declarations for miscellaneous esllib functions (full-screen  
# EGA/VGA color palette functions, and EsIQueryFocus).  
#  
# (C) Copyright Interactive Images, Inc. 1990  
# All Rights Reserved  
#  
#####  
  
function EsISetEgaColor(integer: ColorNumber, integer: ColorValue)  
    returns integer  
    library "esllib"  
function EsIQueryEgaColor(integer: ColorNumber)  
    returns integer  
    library "esllib"  
function EsIQueryFocus()  
    returns string  
    library "esllib"
```

A.8 fileio.inc File

```
#####
#
# Name: FILEIO.INC
#
# Declarations for file input/output routines.
#
# (C) Copyright Interactive Images, Inc. 1989, 1990
# All Rights Reserved
#
#####

subroutine OpenFile( fileid:FileID, string:FileName, string:Fi
                    library "fileio"
subroutine ReadLineNumber( fileid:FileID, integer:LineNumber, string:Target
                    library "fileio"
subroutine ReadNext( fileid:FileID, string:Target )
                    library "fileio"
subroutine ReadRecordAtLine( fileid:FileID, integer:LineNumber, record:EXAMPLE )
                    library "fileio"
subroutine ReadNextRecord( fileid:FileID, record:Example )
                    library "fileio"
subroutine WriteFile( fileid:FileID, string:Source )
                    library "fileio"
subroutine WriteRecord( fileid:FileID, record:Example )
                    library "fileio"
subroutine CloseFile( fileid:FileID )
                    library "fileio"
subroutine SetBufferSize( fileid:FileID, integer:BufferSize )
                    library "fileio"
subroutine SetIndexSize( fileid:FileID, integer:IndexSize )
                    library "fileio"
subroutine SetTabSize( fileid:FileID, integer:TabSize )
                    library "fileio"
subroutine GetError( integer>ErrorLevel, string:Message )
                    library "fileio"

#####
#
# The following declarations are available for compatibility
# with the release 1.0 names of these subroutines. If you wish
# to use the following names, just uncomment the declarations.
# (The library still contains the entry points.)
#
#####

#subroutine Open( fileid:FileID, string:FileName, string:FileMode )
#
#subroutine Close( fileid:FileID )
#subroutine Write( fileid:FileID, string:Source )
```

A.9 mathlib.inc File

```
#####
#
# Name: MATHLIB.INC
#
# Declarations for math library functions.
#
# (C) Copyright Interactive Images, Inc. 1989
# All Rights Reserved
#
#####

function SIN(float:X)           returns float  library "mathlib"
function COS(float:X)           returns float  library "mathlib"
function TAN(float:X)           returns float  library "mathlib"
function ARCCOS(float:X)        returns float  library "mathlib"
function ARCSIN(float:X)        returns float  library "mathlib"
function ARCTAN(float:X)        returns float  library "mathlib"
function ARCTAN2(float:X,
                  float:Y)       returns float  library "mathlib"
function LN(float:X)            returns float  library "mathlib"
function LOG(float:X)           returns float  library "mathlib"
function EXP(float:X)           returns float  library "mathlib"
function ABS(float:X)           returns float  library "mathlib"
function MOD(float:X,
                  float:Y)       returns float  library "mathlib"
function POWER(float:Mantissa,
                  float:Exponential) returns float  library "mathlib"
function SQRT(float:X)          returns float  library "mathlib"
function PI()                   returns float  library "mathlib"
function E()                    returns float  library "mathlib"
function FLOOR(float:X)         returns integer library "mathlib"
function CEIL(float:X)          returns integer library "mathlib"

subroutine MAX(float:Array,
               float:Value)      library "mathlib"
subroutine MAXINT(integer:Array,
                  integer:Value) library "mathlib"
subroutine MIN(float:Array,
               float:Value)      library "mathlib"
subroutine MININT(integer:Array,
                  integer:Value) library "mathlib"
subroutine MEAN(float:Array,
                float:Value)      library "mathlib"
subroutine MEANINT(integer:Array,
                  float:Value)    library "mathlib"
subroutine MEDIAN(float:Array,
                  float:Value)    library "mathlib"
subroutine MEDIANINT(integer:Array,
                    float:Value)  library "mathlib"
subroutine VARIANCE(float:Array,
                    float:Value)  library "mathlib"
subroutine VARIANCEINT(integer:Array,
                      float:Value) library "mathlib"
subroutine STDDEV(float:Array,
                  float:Value)    library "mathlib"
subroutine STDDEVINT(integer:Array,
                    float:Value)  library "mathlib"
subroutine SUM(float:Array,
```

```
float:Value)
subroutine SUMINT(integer:Array,
integer:Value)
```

```
library "mathlib"
library "mathlib"
```

A.10 message.inc File

```
#####
#
# Name: MESSAGE.INC
#
# Definitions to be used in calling ReplyToMessage
#
# (C) Copyright Interactive Images, Inc. 1989, 1990
# All Rights Reserved.
#
#####

#
# ReplyToMessage( Title, MessageText, Options, DefaultOption, Icon)
# returns Reply:String
#
function ReplyToMessage(
    string:Title,
    string:Message,
    integer:Options,
    integer:DefaultOption,
    integer:Icon) returns string library "message"

#
# Title is a string. (This is displayed in the title bar of the message box.)
#
# MessageText is a string.
#
# Options is one of:
    integer constant MessageOK is 0
                      MessageOKCancel is 1
                      MessageRetryCancel is 2
                      MessageAbortRetryIgnore is 3
                      MessageYesNo is 4
                      MessageYesNoCancel is 5

#
# DefaultOption is 1, 2, or 3. (For example, if it's a YesNo message, choose
# option 1 to make Yes the default, option 2 to make No the default and
# option 3 is meaningless.)
#
# Icon is one of:
    integer constant MessageIconQuestion is 16
                      MessageIconExclamation is 32
                      MessageIconAsterisk is 48
                      MessageIconHand is 64
                      MessageNoIcon is 0

                      MessageQuery is 16
                      MessageWarning is 32
                      MessageInformation is 48
                      MessageCritical is 64

#
# Reply will be one of:
#
# "yes"
# "no"
# "abort"
# "retry"
# "cancel"
```



```
# "ignore"  
# "ok"  
#  
# "error"
```

A.11 pmptr.inc File

```
#####  
#  
# Name: PMPTR.INC  
#  
# Presentation Manager system pointers.  
#  
# (C) Copyright Interactive Images, Inc. 1989  
# All Rights Reserved  
#  
#####
```

```
integer constant  
SPTR_ARROW          is 1  # Arrow cursor  
SPTR_TEXT           is 2  # Text entry I-beam  
SPTR_WAIT           is 3  # Hourglass  
SPTR_MOVE           is 5  # Four-way arrow  
SPTR_SIZEWNSE       is 6  # Size border arrow for UL and LR corners  
SPTR_SIZENESW       is 7  # Size border arrow for LL and UR corners  
SPTR_SIZEWE         is 8  # Size border arrow for left and right sides  
SPTR_SIZES          is 9  # Size border arrow for top and bottom  
SPTR_APPICON        is 10 # Large rectangle  
SPTR_ICONINFORMATION is 11 # Hand inside of a stop sign  
SPTR_ICONQUESTION   is 12 # Question mark inside of inverted triangle  
SPTR_ICONERROR      is 13 # Exclamation mark inside of a triangle  
SPTR_ICONWARNING    is 14 # Large asterisk inside of a box  
SPTR_ILLEGAL        is 18 # Diagonal line inside of a circle  
SPTR_FILE           is 19 # Rectangle with UR corner folded down  
SPTR_FOLDER         is 20 # File folder symbol  
SPTR_MULTIFILE      is 21 # Multiple overlapping rectangles  
SPTR_PROGRAM        is 22 # Rectangle, bold on top
```

Appendix B. Specified Operating Environment

B.1 Machine Requirements

EASEL OS/2 EE is designed to execute with the following equipment:

- One of the following system units:
 - IBM Personal Computer XT Model 286
 - IBM Personal Computer AT
 - IBM Personal System/2 Model 30 286, 50, 50Z, 60, 70 386, or 80 386
- 1MB of available memory in addition to the requirements for IBM Operating System/2 Extended Edition.

Note: This does not include requirements for large customer applications. Additional amounts of memory may be required depending on an individual user's needs.

- One diskette drive (high capacity):
 - IBM 1.2MB 5.25-inch diskette drive
 - IBM 1.44MB 3.5-inch diskette drive
- One fixed disk drive with 3.5MB of free space.

Note: This does not include requirements for large or multiple customer data files and applications. Additional fixed disk space may be required depending on an individual user's needs.

- IBM Keyboard: AT, Enhanced 101, or Enhanced 102.
- Mouse pointing device supported by IBM Operating System/2 for the particular system unit.

Note: A mouse pointing device is only required for those applications which have mouse exclusive features. Layout/CUA does have some mouse exclusive features.

- Graphics display adapter and associated display:
 - IBM Enhanced Graphics Adapter with IBM Enhanced Color Display
 - IBM Personal System/2 Display Adapter or IBM Personal System/2 system unit (Model 30 286, 50, 50Z, 60, 70 386, or 80 386) with one of the following displays:
 - IBM Personal System/2 Monochrome Display 8503
 - IBM Personal System/2 Color Display 8512
 - IBM Personal System/2 Color Display 8513
 - IBM Personal System/2 Color Display 8514

- IBM Personal System/2 Display Adapter 8514/A with one of the following displays:
 - IBM Personal System/2 Monochrome Display 8503
 - IBM Personal System/2 Color Display 8512
 - IBM Personal System/2 Color Display 8513
 - IBM Personal System/2 Color Display 8514
- Communications adapters supported by IBM OS/2 Extended Edition Communications Manager for the particular system unit, which provide:
 - LU 6.2 communications (APPC)
 - 3270 communications (EHLLAPI)
 - 5250 communications (EHLLAPI)
 - Asynchronous communications (ACDI)
 - Remote database services (OS/2 Extended Edition Database Manager)

Note: Communications adapters are only required for those applications which have external communications features.

- Audio equipment:
 - IBM Audio Capture and Playback Adapter or IBM Audio Capture and Playback Adapter/A
 - Any of the following audio output devices:
 - System unit built-in speaker (this output device is not recommended)
 - One of the following amplified stereo speaker configurations:
 - Internally amplified stereo speakers
 - Stereo speakers with stereo amplifier
 - Stereo headphones
 - Any of the following audio input devices:
 - Microphone
 - Analog audio source (such as a cassette player or compact disk player)

Note: Audio equipment is only required for those applications which have audio features. Audio features are only available for the IBM Personal System/2.

B.2 Program Requirements

EASEL OS/2 EE is designed to execute on IBM Operating System/2 Extended Edition Version 1.2 or later.

IBM OS/2 Programming Tools and Information Version 1.2 or later is necessary for developing those applications which require customer designed bit-maps, minimize icons, fonts, or dynamic link libraries.

B.3 Graphics Display Considerations

B.3.1 Display Resolution

The following lists the resolution of the graphics display adapters supported by EASEL OS/2 EE:

Graphics Display Adapter	Resolution
IBM Enhanced Graphics Adapter (EGA)	640 x 350
IBM Video Graphics Array (VGA)	640 x 480
IBM Display Adapter 8514/A	1024 x 768

Note: IBM Video Graphics Array (VGA) graphics display adapter is provided as an integrated feature of the IBM Personal System/2 system unit (Models 30 286, 50, 50Z, 60, 70 386, and 80 386) or by the IBM Personal System/2 Display Adapter.

B.3.2 Portability

Although graphics are displayed in screen coordinates (a property common across all monitors), text is displayed in pixels, a physical property of the particular graphics display adapter. A text string looks different on various display resolutions because a pixel represents a different percentage of each display. Use resolution-dependent fonts when you might want portability across displays. (For details, see 5.6.3.2, "Resolution-Dependent Fonts" on page 5-45.)

Appendix C. Fatal Runtime Errors

C.1 Overview

When an error occurs that prevents the EASEL OS/2 EE Runtime Facility from executing or continuing to execute a program, a message box is displayed to the user. This message box contains a code number for the error, a short description of the error encountered, and a push button labeled "OK". The user must select the OK push button to continue. The runtime exits after the push button has been selected.

Generally, these errors cannot be serviced by the user, but must instead be serviced by the application developer or system administrator.

C.2 Error Messages

The following sections describe the EASEL OS/2 EE fatal runtime errors.

The text enclosed in the < > symbols is replaced with the appropriate text for the particular error.

C.2.1 ESL1002

Message: ESL1002: Interface file "< FILENAME >" cannot be found.
Remarks: The requested EASEL OS/2 EE interface file cannot be found, cannot be opened, or cannot be read.

C.2.2 ESL1003

Message: ESL1003: "< FILENAME >" is an old version binary file.
Remarks: The requested EASEL OS/2 EE interface file is incompatible with the current version of the EASEL OS/2 EE runtime facility. The interface must be recompiled before being executed.

C.2.3 ESL1004

Message: ESL1004: "< FILENAME >" is not an EASEL OS/2 EE binary file.
Remarks: The program name requested cannot be executed because it is not an EASEL OS/2 EE binary file.

C.2.4 ESL1005

Message: ESL1005: Interface file "< FILENAME >" is too large for this configuration.
Remarks: The requested EASEL OS/2 EE interface file is larger than the PRGMAX setting in the EASEL OS/2 EE configuration file (CONFIG.ESL).

C.2.5 ESL1006

Message: ESL1006: Missing input file specification.

Remarks: The EASEL OS/2 EE command line does not contain the name of a program to execute.

C.2.6 ESL1007

Message: ESL1007: Global variable mismatch in variable
<VARIABLE_NAME>.

Remarks: When changing to a new program, the new program has declared a global variable or stimulus differently than the previous program.

C.2.7 ESL1008

Message: ESL1008: Maximum global memory size exceeded.

Remarks: The EASEL OS/2 EE program currently executing has requested more global memory than allowed by the GLBMAX setting in the EASEL OS/2 EE configuration file (CONFIG.ESL).

C.2.8 ESL1009

Message: ESL1009: Maximum program memory size exceeded.

Remarks: The EASEL OS/2 EE program currently executing has requested more program memory than allowed by the PRGMAX setting in the EASEL OS/2 EE configuration file (CONFIG.ESL).

C.2.9 ESL1010

Message: ESL1010: Not enough windows available to execute program.

Remarks: The requested EASEL OS/2 EE program cannot execute because not enough windows are available from OS/2. Terminating other running OS/2 Presentation Manager programs may free enough windows to run.

C.2.10 ESL1011

Message: ESL1011: Unknown command line argument "- < ARGUMENT >".

Remarks: The EASEL OS/2 EE command line contains an unrecognized option.

C.2.11 ESL1012

Message: ESL1012: Too many input files specified.

Remarks: The EASEL OS/2 EE command line contains the names of more than one program.

C.2.12 ESL1012

Message: ESL1013: Cannot open error file "- < FILENAME >".

Remarks: The EASEL OS/2 EE error file (as specified by the **-e** option on the command line) cannot be created.

C.2.13 ESL1012

Message: ESL1014: Illegal extension in program name "- < FILENAME >".

Remarks: The program name specified contains a filename extension other than EBI.

Index

-g command line option 11-15

A

- absolute position
 - in coordinate system 5-35
- accelerator keys 10-14
- accel.inc file A-3
- accel.inc include file 10-15
- action bar
 - adding at runtime 10-19
 - changing region 10-19
 - deleting from region 10-20
 - example template 10-17
 - manipulating 10-19
- action bar action
 - model of 10-10
- action bar attribute 3-16, 10-6
- action bar items
 - item classes 10-21
 - manipulating 10-20
 - naming 10-20
- action bar keyword 3-17
- action bar responses 10-27
- action bars
 - adding items to 10-24
 - built-in functions 10-28
 - creating 10-9
- action is statement 11-5, 11-6
- action routines 11-4
 - characteristics of 11-5
- action statement 11-17
 - tracing 24-5
- action statements 6-2
 - in program block 6-12
 - overview 6-23
 - tracing 24-3
 - using 3-7
 - viewports 7-47
 - windows 7-47
- action subroutine
 - definition 3-7
- add action bar command
 - model 10-19
- add statement 6-24, 7-4, 7-10, 8-27
- add to class statement 6-37, 8-27
- add to statement 6-24, 6-25, 7-10, 8-27
- adding items to action bars 10-24
- adding items to pulldowns 10-25
- aligning
 - text 7-45
- ancestry
 - at position attribute 5-20
 - attribute 5-12
 - change statement 6-27
 - changing 6-33
 - coordinate system 5-20
 - deleting children 5-13
 - in clear statement 6-26
 - in desktop 3-19
 - in name conversions 4-55
 - in object references 5-13
 - in string conversions 4-55
 - members of function 6-42
 - screen 5-21
 - textual region 8-2
 - valid parents 5-13
 - visibility 5-8
- ancestry function 4-42, 6-7
 - with object function 6-39
- ancestry of function 4-43, 6-31, 7-59, 8-42
- apio 23-9, 24-5
- apio command
 - tracing 24-5
- APPC support
- APPC support in EASEL OS/2 EE
 - ASCII to EBCDIC
 - conversion 19-22
 - CM considerations 19-100
 - conversation information,
 - getting 19-22
 - DLL messages 19-4, 19-8

- APPC support in EASEL OS/2 EE
 - (continued)
 - EBCDIC to ASCII
 - conversion 19-22
 - events, responding to 19-4, 19-8
 - overview 19-2
 - profile subroutines 19-13
 - profiles, description 19-12
 - reserved constants 19-98
 - subroutine list 19-11
 - APPCAcceptStart 19-26
 - APPCConvertA2E() 19-28
 - APPCConvertE2A() 19-31
 - APPCEnd() 19-34
 - APPCFlush() 19-36
 - APPCGetAcceptProfileNames() 19-41
 - APPCGetAcceptProfile() 19-38
 - APPCGetDLLData() 19-43
 - APPCGetInfo() 19-46
 - APPCGetInitProfileNames() 19-53
 - APPCGetInitProfile() 19-50
 - APPCGetString() 19-55
 - APPCGetSystemError() 19-57
 - APPCReadyToReceive() 19-61
 - APPCReceiveNotify() 19-63
 - APPCRequestConfirm() 19-65
 - APPCRequestToSend() 19-67
 - APPCSendConfirmed() 19-69
 - APPCSendDLLData() 19-71
 - APPCSendError() 19-75
 - APPCSendString() 19-77
 - APPCSetAcceptProfile() 19-81
 - APPCSetInitProfile() 19-84
 - APPCStart() 19-88
 - APPCTestReqToSend() 19-91
 - append statement 4-21
 - application
 - as global entity 14-2
 - communications for 14-5
 - creating 3-5
 - distributing 2-8
 - escape characters 14-7
 - event-driven 2-5
 - local 14-2
 - match clause for 14-9
 - overview 14-1

- application (continued)
 - remote 14-2
 - remote, characteristics 14-5, 14-21
 - responding to 14-2
 - starting 14-3
 - starting and stopping 14-3
 - starting local 14-4
 - starting, remote 14-5
 - tracing communication 24-5
 - writing 2-5
- application Environment
 - Declaration 14-3, 14-7
- application statement 14-2
- applications
 - as global entity 11-14
 - match clause for 14-8
 - sending escape sequences 4-15
 - windowed vs fullscreen 3-19
- application, event-driven 2-5
- arc
 - drawing 7-24, 7-26
 - moving in 7-28
 - specified degrees 7-24
 - specified endpoint 7-26
- array
 - defining 4-37
 - dimensions 4-37
 - elements 4-38
 - global 4-39
 - global string variable 11-33
 - initializing 4-37, 4-38
 - memory 4-40
 - referencing elements 4-39
 - resizing 4-37, 4-40
 - subscript 4-37
 - value 4-38
- ASCII
 - file, coloring text in 8-5, 8-58
 - file, default colors in 8-59
 - file, inserting 8-20
 - text file, reading 8-33
- ASCII files
 - printing 22-5
- ASCII to EBCDIC conversion in
 - APPC 19-22

- asc1115 keyword 5-46
- asc1521 keyword 5-46
- asc37 keyword 5-46
- asc510 keyword 5-46
- asc59 keyword 5-46
- asc713 keyword 5-46
- asc79 keyword 5-46
- asc814 keyword 5-46
- asc88 keyword 5-46
- at default position attribute 7-4
- at position attribute 6-26, 7-3
- attribute
 - action bar 10-6
 - border 10-4
 - maximize 10-5
 - minimize 10-4
 - no scale 10-7
 - pointer 10-7
 - primary 3-19
 - priority 7-73
 - scroll 10-6
 - system menu 10-5
 - title bar 10-4
 - visibility 7-73
- attributes 5-3
 - ancestry 5-12
 - color 5-8
 - defining 3-17
 - frame related 3-17
 - frame style 3-16
 - parameter 5-15
 - priority 5-14
 - selectability 5-5
 - visibility 5-8
- AUDClose() 18-9
- AUDErase() 18-10
- audio files
 - creating 18-21
 - deleting 18-10
 - description 18-2
 - errors 18-5, 18-13, 18-14, 18-21
 - size considerations 18-3, 18-22
 - specifying path of 18-11
- audio support
 - all, list of 18-6
 - AUDClose() 18-9
 - audio support (*continued*)
 - AUDErase() 18-10
 - AUDOpen() 18-11
 - AUDPause() 18-12
 - AUDPlay() 18-13
 - AUDQueryBusy() 18-17
 - AUDQueryElapsedTime() 18-19
 - AUDQueryPlayingTime() 18-20
 - AUDRecord() 18-21
 - AUDResume() 18-25
 - AUDSetBalance() 18-26
 - AUDSetVolume() 18-28
 - AUDStop() 18-29
 - audio support in EASEL OS/2 EE
 - audio files description 18-2
 - audio utility 18-3
 - DLL messages 18-4
 - events, responding to 18-4
 - IBM Audio Capture and Playback Adapter/A 18-2, 18-13, 18-14, 18-22, 18-23
 - integer constants 18-5, 18-7
 - overview 18-2
 - reserved constants 18-7, 18-33
 - sample program 18-31
 - sound output 18-2
 - sound source 18-2, 18-22
 - speakers, external 18-2, 18-13, 18-14, 18-26, 18-28
 - speaker, workstation 18-2, 18-13, 18-14, 18-26, 18-28
 - subroutine list 18-6
 - audio utility program 18-3
 - AUDIO.EBI 18-3
 - AUDOpen() 18-11
 - AUDPause() 18-12
 - AUDPlay() 18-13
 - AUDQueryBusy() 18-17
 - AUDQueryElapsedTime() 18-19
 - AUDQueryPlayingTime() 18-20
 - AUDRecord() 18-21
 - AUDResume() 18-25
 - AUDSetBalance() 18-26
 - AUDSetVolume() 18-28
 - AUDStop() 18-29

- automatic interface, APPC
 - description 19-14
 - overview 19-3
 - sample program 19-93
 - subroutines 19-15
- A3270.EXE application 20-2
- A5250.EXE application 21-2

B

- b 23-3
- background at function 4-43, 6-30, 8-42, 8-47
- background attribute 7-4
- background color 5-10
 - textual region 6-30
- background of function 4-43, 6-30, 7-59, 8-42
- backgroundspace character 8-16, 8-34
 - overwriting 8-22
- BEEPFREQ 15-14
- begin statement 6-11
- BGraph module
 - GAutoScaleX 16-20
 - GAutoScaleY 16-20
 - GAutoXTitle 16-19
 - GAutoYTitle 16-19
 - GAxisTitleFont 16-18
 - GBottomX 16-20
 - GBottomY 16-20
 - GColor 16-28
 - GDataElements 16-8
 - GDataSets 16-8
 - GDataX 16-6
 - GDataY 16-6
 - GDecimalPoint 16-8
 - GFootnote 16-27
 - GFootnoteColor 16-28
 - GFootnoteFont 16-27
 - GLabels 16-22
 - GLegend 16-25
 - GLegendColor 16-26
 - GLegendFont 16-26
 - GLegendOn 16-25
 - GLineWidth 16-33
 - GMarkerColor 16-32

- BGraph module (*continued*)
 - GMarkerLines 16-31
 - GMarkerLineWidth 16-33
 - GMaxDataElements 16-23
 - GMissingDataValue 16-7
 - GMovables 16-35
 - GPatterns 16-34
 - GPieDataSet 16-12
 - GPieLabels 16-21
 - GraphOnLast 16-35
 - GraphOverlap 16-10
 - GraphPie 16-11
 - Graph3D 16-10
 - GRedNegativeBars 16-29
 - GRegion 16-3
 - GridX 16-30
 - GridY 16-30
 - GStepLineSet 16-14
 - GTickFont 16-23
 - GTitle 16-16
 - GTitleColor 16-17
 - GTopX 16-20
 - GTopY 16-20
 - GXMarkers 16-31
 - GXTickIncrement 16-24
 - GxTitle 16-17
 - GxTitleColor 16-19
 - GYMarkers 16-31
 - GYTickIncrement 16-24
 - GyTitleColor 16-19
 - G3DOutline 16-11
 - XNumTicks 16-23
 - XTicks 16-23
 - YNumTicks 16-23
 - YTicks 16-23
- binddb command 17-6
- bindfile command 17-12
- blank 14-12
- blank in match clause 14-9
- blanks
 - in program 4-2
- block definition
 - column 8-12
 - line 8-12
 - thru 8-12

block See Block (Text)

text 8-12

block statement 11-17

block (Program) 14-14

active 6-10

active, in nested 6-15

definition 6-11

entering, responding to 6-20

exiting 6-13, 6-14, 6-19

exiting, responding to 6-16

guarded 6-12, 6-14, 6-18

guarded, exiting 6-19

identifier 6-11

inactive 6-10

loop in 11-26

nested 6-10, 6-12, 6-15

overview 6-10

response to char

statement 6-21

response to start

statement 6-20

resumable 6-12

block (Text)

blanks 8-25

copying contents of 8-43, 8-45

emphasizing 8-30

in color header 8-60

in colored textual regions 8-55

overwriting 8-23, 8-25

removing 8-28

boolean

value 11-18

BOOPFREQ 15-14

border

in shape 7-29

border attribute 7-4, 10-4

size or standard 3-16

border of function 4-43, 6-30, 7-60,
8-42

border statement 7-29

bottom of function 4-43, 7-55, 8-41

boundary

in circle 7-21

in ellipse 7-21

in shape 7-20

specification 7-20

box

color of 7-14

colored 7-14

hollow 7-14

solid 7-14

statement 7-13

box statement 7-9

color keyword 7-13

solid keyword 7-13

buffer, examining

3270 20-14

5250 21-14

built-in functions 4-42, 7-54

action inquiry 4-44

in item inquiries 10-28

in item response 10-29

item inquiry 4-44

object attributes 7-59

object inquiry 4-43

response inquiry 4-42

special inquiry 4-45

textual region 8-41

Business Graphics

action routines 16-5

axis boundaries 16-20

Axis Titles 16-17

colors 16-28

creating 16-3

customizing 16-15

footnotes 16-27

graph titles 16-16

illustrations 16-41

introduction 16-2

legends 16-25

line graph 16-41

overlapping bar graph 16-10,
16-45

pie graph 16-11, 16-43

pie wedge labels 16-21

public file 16-51

required procedures 16-3

rules for 16-10

sample program 16-38

scatter graph 16-44

stacked bar graph 16-46

stacked line graph 16-42

Business Graphics (*continued*)

- standard bar graph 16-44
 - standard line graph 16-41
 - stepped line graph 16-14, 16-42
 - superimposing two graphs 16-35
 - surface line graph 16-43
 - 3-D bar graph 16-10
 - 3-D overlapping bar graph
 - three dimension ov 16-46
 - 3-D stacked bar graph 16-47
 - three dimensional bar graph 16-10
 - three dimension o 16-11
 - three dimension ov 16-46
 - three-dimension 16-45
 - tick mark labels 16-21
 - variables 16-10
 - variables, summary of 16-48
- buttons
- defining 10-13
- by keyword 5-35

C

- C routines 25-3
- cache
 - in image regions 7-65
 - no cache 7-66
 - work area 7-66
- cache image 7-65
- cache viewport 7-66
- Canadian-French code page 4-4
- caps814 keyword 5-46
- centering
 - centering 7-45
- change graphics statement 5-12, 6-24, 6-27, 6-29, 7-10, 8-27
- change item statement
 - model 10-23
- change position statement 5-28, 5-29, 6-31, 7-50, 8-27
- change priority statement 5-15, 6-31
- change size statement 5-28, 5-30, 7-48

- change statement 6-24, 6-27, 6-29, 7-10, 8-27
- change text action statement 3-18
- change text to action
 - model of 10-8
- change to program
 - transfer control 11-11
- change to program command 3-21
- change to program statement 4-39, 6-3, 11-2, 11-10, 11-11, 15-2, 15-7
 - with global arrays in programs 11-15
 - with global entities 11-15
 - with global variables 11-33
- change to statement 7-4
- change window position 7-52
- change window position statement 5-25
- change window size
 - statement 5-25
- changes
 - enhancements 3-3
 - features, new 3-2
 - summary 3-2
- changing
 - ancestry 6-31, 6-33
 - attributes 6-28
 - border 6-30
 - color 6-29
 - parameter 6-31
 - position 6-31
 - priority 6-31
 - selectability 6-28
 - size 6-31
 - visibility 6-28
- changing position
 - window 7-52
- changing size
 - window. 7-51
- char
 - in background at statement 8-47
 - in foreground at statement 8-47
- character cell 5-43
 - intercell gap 5-43
- character set 4-2

- check boxes 9-20
- check keyword 10-22
- checked in group function 4-43
- choice definition
 - model of 10-12
- circle
 - color of 7-16
 - colored 7-16
 - solid 7-16
 - solid or hollow 7-16
 - statement 7-15
- circle statement 7-9, 7-20
 - radius keyword 7-15
- class
 - actions for 6-36
 - adding object to 6-37
 - ancestry 6-42
 - ancestry function 6-40
 - changing membership 6-38
 - defining 6-34
 - definition 3-7
 - deleting object from 6-38
 - description 6-34
 - in action statement 6-23
 - members of function 6-41
 - parameter function 6-40
 - responding to 6-35
- clear graphics statement 5-12, 6-24, 6-26, 8-27
- clear statement 6-24, 6-26, 8-27
- clipboard function 4-45
- close statement 15-7
- close subroutine 13-31
 - error messages 13-35
- code page support 4-4
- codepage
 - installation 1-4
- col in match clause 14-9, 14-10
- color
 - additional set 5-9
 - background 5-10
 - border 5-10
 - changing 6-29
 - controlling 3-20
 - ellipse 7-17
 - foregrounds 5-10
 - color (*continued*)
 - graphical regions 7-7
 - in drawing statements 5-11
 - in regions 5-10
 - lines 7-12
 - of a box 7-14
 - of a line 7-12
 - of a polygon 7-15
 - of a shape 7-29
 - of background 8-47
 - of circle 7-16
 - of foreground 8-47
 - of pattern 7-35
 - of text 7-46, 8-5
 - of wedge 7-19
 - polygon 7-15
 - standard set 5-9
 - text segments 8-23
 - values 5-9
 - color keyword 5-9, 5-10
 - color library
 - EslQueryEgaColor function 13-4
 - EslSetEgaColor function 13-4
 - color numbers 5-9
 - color values vii
 - color-attributed file 8-6
 - reading into a textual region 8-34
 - writing to 8-32
 - colored textual region
 - graphics boards for 8-54
 - colored textual region statement 8-54
 - colored textual regions
 - color 8-54
 - coloring
 - text 8-55
 - column
 - in background at statement 8-47
 - in foreground at statement 8-47
 - column size of function 4-43, 8-46
 - combination boxes 9-15
 - command line options
 - apio 23-9
 - debug 23-10
 - e 23-10

command line options (*continued*)

- f 23-10
- g 23-10
- m 23-4, 23-10
- nb 23-10
- terp 23-10
- apio
- b 23-3
- d
- debug 23-3
- e 23-3
- f 23-4
- fullscreen 23-4
- g 23-4
- lx 23-4
- m
- nb
- SQL 17-30
- terp
- comment
 - in color header 8-60, 8-61
- comments
 - in ASCII text file 8-60
 - in program 4-3
- communicating with host
 - 3270 20-15
 - 5250 21-49
- communications 24-5
 - characteristics 14-5
 - with application 14-7
- communications port
 - characteristics 14-21
- compile command 3-12, 3-13
- compile-time value 4-11
- COMPILE.CMD 3-19
- COMPILE.CMD OS/2 Command File 23-3
- compiling 3-12
- complement mode 15-15
- components
 - of an object 5-3
- conditional statements
 - Nested 11-20
- configuration commands 17-7
- configuration options
 - BEEPFREQ 23-5

configuration options (*continued*)

- BOOPFREQ 23-5
- CODEPAGE 23-6
- ESLERR 23-6
- EXETYPE 23-6
- FontCNT 23-6
- FontPATH 23-6
- GLBMAX 23-6
- GLBSIZE 23-6
- HLDELAY 23-6
- INCPATH 23-6
- INCTYPE 23-6
- KBDTSBWID 23-7
- LOCALDLL 23-7
- MODPATH 23-7
- POINTER 23-7
- PRGMAX 23-7
- PRGSIZE 23-7
- REMOTE 23-7
- REMOVEDLL 23-7
- SLEEPMAX 23-7
- TABWIDTH 23-7
- TEXTPATH 23-7
- TEXTTYPE 23-8
- XMAX 23-8
- YMAX 23-8
- configuring CM for EASEL OS/2 EE APPC 19-100
- CONFIG.ESL 1-4
- CONFIG.ESL file 3-13, 5-42, 11-8, 23-5
 - editing during install 3-13
- CONFIG.SYS 1-4
- CONFIG.SYS file 15-14, 15-15
 - CODEPAGE statement 4-4
 - COUNTRY statement 4-4
 - DEVINFO statement 4-4
- confirmation processing, APPC
 - description 19-16
 - overview 19-5
 - subroutines 19-17
- Connect 20-19
- connecting
 - 3270 20-14
 - 5250 21-14

- constant 4-16
- constant statement 4-16
- constant statement(s) 3-6
- controlled interface, APPC
 - description 19-18
 - overview 19-4
 - sample program 19-95
 - subroutines 19-21
- controlling program
 - Timing 11-29
- conversion
 - to uppercase 5-46
- conversions
 - boolean/string 4-54
 - integer/floating point 4-54
 - string/floating point 4-53
 - string/integer 4-53
 - string/name 4-55
- coordinate
 - of selection 6-9
- coordinate system
 - ancestor 5-20
 - graphical object 5-18
 - object 5-18
 - of regions 5-22
 - of screen 5-16
 - of viewport 5-22
 - of window 5-22
 - textual region 5-20
- coordinates
 - of a pattern 7-34
- copy statement 4-20
- courier font 5-47
- creating
 - action bars 10-9
- CText module 8-58, 8-60
- CTR_File variable 8-58, 8-61
- CTR_Header variable 8-58, 8-61
- CTR_Read action subroutine 8-58
- CTR_Region variable 8-58, 8-61
- cursor
 - graphics 5-33
 - in drawing statement 5-32
 - in objects 5-32
 - mouse 15-2
 - text 5-35

- cursor movement
 - in box statement 5-33
 - in circle statement 5-33
 - in draw statement 5-33
 - in move statement 5-33
- cursors
 - graphics 7-10
 - moving 7-10

D

- databits 14-22
- date
 - library functions 13-7
- date function 4-45
- date library
 - DaySpan function 13-7
 - LocalDate function 13-8
 - LocalTime function 13-8
 - Validate function 13-9
 - WeekDay function 13-10
- date library functions 13-7
- datelib.inc file A-4
- debug 23-3
- debugging 24-1
- Declaring a stimulus 14-29
- deemphasize statement 8-30
- default
 - in color header 8-59, 8-61
- DefineWatch 20-35
- defining template
 - action bar 10-10
- delete action bar command
 - model 10-20
- delete from class statement 6-38, 8-27
- delete item statement
 - model 10-22, 10-26
- delete statement 6-24, 6-28, 8-27
- deleting from classes 6-26, 6-27
- deleting items from regions 10-26
- desktop
 - ancestor 5-12, 5-21
 - screen size 5-18
- desktop keyword 3-19

- development process 2-7
- device units 3-16
- DEVINFO = KBD statement 4-6
- dialog box
 - sample code 9-24
- dialog box definition
 - model 9-2
- dialog boxes 9-2
 - sample program 9-24
- dialog control objects 9-7
- Dialog region 9-5
- dialog units 3-16
- digitized audio files 18-2
- direct commands 17-3
- directory
 - created 1-7
- disable statement 5-7, 6-28
- disabled keyword 5-5
- Disconnect 20-20
- disconnecting
 - 3270 20-14
 - 5250 21-14
- display 5-16
- display command 17-22
- display resolution B-5
- displaying
 - text 7-44
- DLL declaration, audio 18-4
- DLL declaration, in APPC 19-5
- DLL messages, APPC 19-4, 19-8
- DLL messages, audio 18-4
- draw arc statement 7-19, 7-20, 7-24
 - center 7-24
 - clockwise keyword 7-25, 7-26
 - counter clockwise
 - keyword 7-25, 7-26
- draw statement 5-19, 7-9, 7-11, 7-14, 7-20
 - color keyword 7-11
- drawing
 - line 7-11
- drawing statement
 - description 5-4
 - for text 7-44
 - graphical objects 7-9
 - graphics cursor 5-33

- drawing statement (*continued*)
 - placement 7-9
 - set of 7-30
 - text cursor 5-35, 5-37
- drawing statements
 - programs 8-7
 - textual region 8-2, 8-7
 - where to place 8-7
- drop 14-16
- drop keyword 15-7
- drop-down combination box 9-15
- dynamic commands 17-26

E

- e 23-3
- EASEL OS/2 EE
 - application, creating 3-5
 - benefits 2-3
 - compiles 3-15
 - compiling 3-12
 - development process 2-7
 - executes 3-15
 - executing 3-12
 - fonts 5-45
 - front-end application 2-5
 - installing 1-4
 - introduction 2-2
 - library functions 13-4
 - program construction 3-6
 - programming 3-1
 - sample program 3-9
 - screen 5-16
- EASEL OS/2 EE library
 - functions 13-4
- EASEL OS/2 EE program
 - exiting 11-9
- EASEL OS/2 EE regions
 - plotting 22-2
 - printing 22-2
- EASEL OS/2 EE sample programs
 - textual regions 8-49
 - using CText module 8-62
- EASEL.CMD file 24-5
- EASEL.CMD OS/2 Command
 - File 23-9

- EBCDIC to ASCII conversion in
 - APPC 19-22
- EGA
 - fullscreen colors 5-9
- egacolor.inc
 - include file 13-4
- egacolor.inc include file 13-4, A-5
- EGA/VGA
 - color values 13-5
 - ENTRY color values 13-5
- EGA/VGA color values 13-5
- elements
 - referencing in array 4-39
- ellipse
 - color of 7-17
 - colored 7-17
 - solid 7-17
 - solid or hollow 7-17
 - statement 7-16
- ellipse statement 7-9, 7-16, 7-20
- emphasize statement 8-29
- emphasizing
 - in text 8-29
 - segment 8-29
- EmulateAndWatch 20-53
- Emulate3270 20-52
- enable statement 5-7, 6-28
- enabled 6-4, 6-6
- enabled keyword 3-9, 5-5
- end loop statement 11-21
- end shape statement 7-20
- end statement 6-11
- EnterString 20-49
- entity
 - local 11-14
- entry field definition
 - model 9-8
- entry fields
 - multiple-line 9-9
 - single-line 9-8
- entry groups of dialog controls 9-7
- environment declaration
 - application 3-6
 - font 3-6
 - module 3-6
- environment files 23-1
- environmental definition 14-2
- eparse command 3-19
- error command 17-9
- error handling 21-60
 - SQL 17-27
 - 3270 20-64
 - 5250 21-60
- error messages C-1
 - fatal runtime C-1
 - for close subroutine 13-35
 - input subroutines 13-40
 - open subroutine 13-34
 - output subroutines 13-43
 - parameter adjustment subrou-
tines 13-46
 - plotting 22-4
 - printing 22-7
 - SQL 17-32
 - 3270 20-66
 - 5250 21-62
- error messages for math subrou-
tines 13-13
- error status variables
 - 3270 20-64
 - 5250 21-60
- errorlevel function 4-45
- errorlog 24-2
 - sending values to 24-3
 - text in 24-3, 24-4, 24-5
- errorlog file
 - using 24-2
- errorlog keyword
 - in send statement 24-4
- escape sequences
 - in files to be read in 8-34
 - in string literals 4-14
 - inserted strings 8-16
 - overwriting 8-22
 - quotation mark 4-15
 - special keys 15-9
- eslapcc.inc include files A-6
- esludio.inc include files A-12
- esllib.inc include files A-14
- ESLPR.EXE utility
 - header keyword 22-6

ESLPR.EXE utility (*continued*)

- lpt1 keyword 22-6
- lpt2 keyword 22-6
- lpt3 keyword 22-6
- pagesize keyword 22-6
- ESL1002 C-3
- ESL1003 C-3
- ESL1004 C-3
- ESL1005 C-3
- ESL1006 C-4
- ESL1007 C-4
- ESL1008 C-4
- ESL1009 C-4
- ESL1010 C-5
- ESL1011 C-5
- ESL1012 C-5
- evenpar 14-21
- event notifications, APPC 19-4, 19-8
- event notifications, audio 18-4
- eventnumber built-in function 14-30
- eventnumber function 4-42
- eventparam built-in function 14-30
- eventparam function 4-42
- executing 3-12
- exists function 4-43, 7-60, 8-42
- exit
 - from a loop 11-24
- exit action statement 11-9
- exit command 17-20
- exit statement 11-2, 11-9, 15-7
- expression 4-47
 - compile-time 4-47
 - compound 4-47
- extract from statement 4-22, 14-14
 - column 4-32
 - float 4-30
 - last 4-26
 - marker 4-24
 - number 4-29
 - skip clause 4-22, 4-24
 - take clause 4-22, 4-24
 - text 4-31
 - with textual built-in function 8-46

extract from statement (*continued*)

- word 4-27

F

- f 23-4
- fatal runtime errors C-1
 - ESL1002 C-3
 - ESL1003 C-3
 - ESL1004 C-3
 - ESL1005 C-3
 - ESL1006 C-4
 - ESL1007 C-4
 - ESL1008 C-4
 - ESL1009 C-4
 - ESL1010 C-5
 - ESL1011 C-5
 - ESL1012 C-5
- features, new 3-2
- fields utility 20-8
 - retriving information 20-8
 - sample output 20-10, 21-11
 - 3270 20-2
 - 5250 21-3
- file contents
 - of Image regions 7-68
- file i/o library 13-26
 - subroutines 13-26
- file i/o subroutines
 - ASCII files 13-26
 - parameter types 13-27
 - record definition 13-27
- fileio.inc file A-15
- files
 - installation 1-7
- find statement
 - column keyword 8-34
 - found function 4-44
 - from keyword 8-34
 - line keyword 8-34
- FindLastNonBlankLine 20-29
- fixed-cell font 5-45
- flag
 - fullscreen 3-19
- font
 - and ancestry 5-43

- font (*continued*)
 - and screen size statement 5-45
 - attributes 5-49
 - character cells 5-43
 - character descenders 5-44
 - defining 5-48
 - EASEL OS/2 EE types 5-45
 - environment declaration 3-6
 - fixed-cell 5-44, 5-45, 5-46
 - fixed-width 5-47
 - image 5-49
 - in pattern 7-36
 - in textual region 5-50
 - inheritance 5-43
 - maximum in program 5-42
 - monospaced 5-47
 - names and sizes 5-45
 - of text 7-46
 - pixels 5-43
 - PM types 5-47
 - portability 5-44
 - proportional 5-47
 - resolution-dependent 5-44, 5-45, 5-46
 - scalable 5-50
 - uppercase 5-46
- font attribute 5-50
- font is statement 5-48
- font statement 5-42
- FONTCNT option 5-42
- fonts
 - textual region 8-3, 8-6
- for each selected line statement
 - model 9-13
- foreground at function 4-43, 6-30, 8-42, 8-47
- foreground color 5-8, 5-10
 - textual region 6-30
- foreground of function 4-43, 6-30, 7-59, 8-42
- found function 4-44, 8-34, 8-43, 8-48
- frame attributes 3-17
- frame oriented syntax 3-16
- frame related attributes 3-17

- frames
 - defining 3-17
- freeSize function 4-45, 11-34, 11-35
- front-end application 2-5
- fs command 17-14
- full-duplex conversations 19-2
- fullscreen 23-4
- fullscreen flag 3-19
- fullscreen vs windowed
 - applications 3-19
- function declaration 11-2
- functions 11-4
 - writing 25-3

G

- g 23-4
- g command line option 11-34
- GAutoScaleX 16-20
- GAutoScaleY 16-20
- GAutoXTitle 16-19
- GAutoYTitle 16-19
- GAxisTitleFont 16-18
- GBottomX 16-20
- GBottomY 16-20
- GColor 16-28
- GDataElements 16-8
- GDataSets 16-8
- GDataX 16-6
- GDataY 16-6
- GDecimalPoint 16-8
- GetCursorPosition 20-25
- GetFieldAttributes 20-24
- GetFieldLength 20-23
- GetFieldNumber 20-26
- GetFieldPosition 20-27
- GetSessionInfo 20-62
- GetSessionStatus 20-60
- GetXstatus 20-57
- GFootnote 16-27
- GFootnoteColor 16-28
- GFootnoteFont 16-27
- GLabels 16-22
- GLegend 16-25
- GLegendColor 16-26

- GLegendFont 16-26
- GLegendOn 16-25
- GLineWidth 16-33
- global entity 11-14
- global items
 - defining 10-14
- global memory 11-33
- GMarkerColor 16-32
- GMarkerLines 16-31
- GMarkerLineWidth 16-33
- GMaxDataElements 16-23
- GMissingDataValue 16-7
- GMovables 16-35
- GPatterns 16-34
- GPieDataSet 16-12
- GPieLabels 16-21
- graphical object
 - coordinate system 5-18
 - graphics cursor 5-33
- graphical objects 7-2
 - built-in functions for 7-54
 - built-in functions 7-54
 - drawing statements 7-9
- graphical region
 - ancestry 7-6, 7-7
 - and ancestry 7-6
 - attributes 7-5
 - background 7-7
 - color 7-7
 - coordinate system 5-18
 - default attributes 7-5
 - defining 7-4
 - foreground 7-8
 - graphics cursor 5-33
 - position 7-6
 - size 7-6
 - size and position of 7-6
 - statement 7-4
 - text, image 7-44
 - text, outline 7-45
 - viewport 7-47, 8-36
 - viewport, size of 7-48
 - window 7-47, 8-36
 - window, changing position 7-52
 - window, changing size of 7-51
- graphical region definition
 - border keyword 7-5
 - color keyword 7-5
 - default keyword 7-5
 - disabled keyword 7-5
 - enabled keyword 7-5
 - foreground keyword 7-5
 - invisible keyword 7-5
 - parameter keyword 7-5
 - position keyword 7-5
 - visible keyword 7-5
- graphical region statement 7-4
 - border keyword 10-3
- graphical regions
 - viewports 7-47
 - windows 7-47
- graphical region, changing position of
 - viewport 7-50
- graphics boards
 - EGA 8-54
 - VGA 8-54
- graphics cursor 5-33
- GraphOnLast 16-35
- GraphOverlap 16-10
- GraphPie 16-11
- Graph3D 16-10
- GRedNegativeBars 16-29
- GRegion 16-3
- GridX 16-30
- GridY 16-30
- group box definition
 - size in 9-23
- group boxes 9-23
- GStepLineSet 16-14
- GTickFont 16-23
- GTitle 16-16
- GTitleColor 16-17
- GTitleFont 16-16
- GTopX 16-20
- GTopY 16-20
- guarded block
 - definition 6-11
 - description 6-18
- GXMarkers 16-31

- GXTickIncrement 16-24
- GxTitle 16-17
- GxTitleColor 16-19
- GYMarkers 16-31
- GyTickIncrement 16-24
- GyTitleColor 16-19
- G3DOutline 16-11

H

- half-duplex conversations 19-2
- handle of 7-61
- handle of function 4-43, 8-42
- hardcopy support 22-2
- helv font 5-47
- highlight statement 15-14
- HLDELAY 15-15
- hold 14-16
- horizontal scroll bar keyword 3-17

I

Identifier

- converting to string 4-55
- description 4-8
- predefined 4-57

identifiers vii

if statement 11-2

- if/then/else statement 11-17, 11-18, 11-21

ignore tab

- in multiline entry field
- definition 9-10

image font 5-49

image region definition

- color keyword 7-64
- default keyword 7-64
- file keyword 7-68
- foreground keyword 7-64
- model of 7-64
- no scale specification 7-68
- position keyword 7-64
- preserve foreground
- statement 7-67

Image regions 7-63

- background colors 7-65

Image regions (*continued*)

- bitmap file formats 7-63
- cache specifications 7-65
- defining 7-64
- file contents of 7-68
- foreground colors 7-65
- operations 7-64

in line statement

- deselect keyword 9-14
- select keyword 9-14

include file 13-2

include files A-1

- accel.inc A-3
- APPC, overview 19-5
- audio, overview 18-3
- datelib.inc A-4
- egacolor.inc A-5
- eslappc.h 19-43, 19-71
- eslappc.inc 19-5, A-6
- eslaudio.inc 18-3, 18-7, 18-9, 18-10, 18-11, 18-12, 18-13, 18-17, 18-19, 18-20, 18-21, 18-25, 18-26, 18-28, 18-29, A-12

- esllib.inc A-14

- fileio.inc A-15

- mathlib.inc A-16

- message.inc A-18

- pmptr.inc A-20

include statement 3-6, 11-2, 11-8

- ASCII text file 11-8

- include statement 11-8

- nested files 11-8

INCTYPE 11-8

info command 17-11

Init3270 20-17

input

- subroutines 13-35

- input function 4-22, 4-42, 6-7, 6-8, 14-14, 15-9

input subroutines 13-35

- error messages for 13-40

inquiry

- functions for 7-54

- position 7-54

- size 7-54

- inquiry commands 17-21
- insert statement 8-14, 8-44, 8-45
 - colored keyword 8-14
 - file keyword 8-14
 - from keyword 8-14
 - ioerror function 4-44
 - thru keyword 8-14
- inserting
 - block (Text) 8-18
 - color-attributed file 8-20
 - segment 8-17
 - string 8-15
 - text 8-14
- installation 1-1, 1-13
 - Development System 1-4
 - directory 1-7
 - diskettes 1-6
 - error messages 1-13
 - files, added 1-7
 - Runtime Facility 1-5
- installing EASEL OS/2 EE 1-4
- integer constants
 - in APPC support 19-5, 19-8, 19-10, 19-23, 19-24, 19-43, 19-57, 19-58, 19-59, 19-71, 19-98
 - in audio support 18-5, 18-7, 18-17
- interrupt response 6-19
- introduction to audio support 18-2
- introduction to EASEL OS/2 EE
 - APPC support 19-2
- invisible keyword 5-8
- invoke action statement 11-10
- ioerror function 4-44, 8-20, 8-43, 8-61
- is checked function 4-43, 4-44
- is enabled function 4-44
- item class definition
 - model 10-21
- item keyword 10-22
- item parameters 10-16
- item reference keyword
 - model of 10-14
- items
 - changing text of 10-23
 - checking and unchecking 10-22

- items (*continued*)
 - enabling and disabling 10-22

K

- KBDTABWID keyword 15-12
- key
 - attributes 7-2
 - color 7-4
 - default attributes 7-2
- key statement 7-2
 - color keyword 5-3, 7-2
 - default keyword 5-3, 7-2
 - disabled keyword 7-2
 - enabled keyword 7-2
 - invisible keyword 7-2
 - parameter keyword 7-2
 - position keyword 5-3, 7-2
 - priority keyword 7-2
 - visible keyword 7-2
- keyboard 14-16
 - layout 4-6
 - supported 4-5
- keyboard mapping table 15-9
- keys
 - objects 7-2
- Keyword 4-8

L

- language
 - supported 4-4
- large keyword 5-46
- Layout.CUA
 - pulldown choice 12-25
- Layout/CUA 12-2, 12-4, 12-5, 12-7, 12-8
 - action bar 12-14
 - action bar choice 12-25
 - adding objects 12-7
 - aligning objects 12-13
 - alignment options 12-13
 - automatic dialog box
 - calling 12-23
 - check box 12-31
 - clipboard 12-16

Layout/CUA (*continued*)

- code generation 12-21
 - combination box 12-36
 - creating interface 12-7
 - deleting objects 12-12
 - dialog box subroutines 12-21
 - dialog boxes 12-8
 - drop-down combination box 12-36
 - drop-down list 12-36
 - entry field, multiline 12-35
 - entry filed 12-33
 - group box 12-34
 - help system 12-5
 - hiding/showing windows 12-14
 - introduction 12-2
 - list box 12-32
 - moving objects 12-12
 - multiline entry field 12-35
 - object information 12-13, 12-24
 - primary window 12-4, 12-24
 - pulldown 12-14
 - pulldown choices 12-23
 - push button 12-29
 - push buttons 12-23
 - query subroutine 12-22
 - radio button 12-30
 - resizing objects 12-12
 - secondary window 12-8, 12-27
 - selection fields 12-15
 - setup subroutine 12-21
 - static box 12-34
 - undo 12-14
 - using 12-4
- lcaps keyword 5-46
- leave block statement 6-10, 6-13, 6-14, 6-19
- leave loop statement 6-19, 11-2, 11-17, 11-21, 11-24, 11-25
- left of function 4-43, 7-55, 8-41
- length of function 4-19, 4-45
- Libraries
- stimulus 14-29
 - writing 25-1, 25-3
- library
- functions 13-3

library (*continued*)

- subroutines 13-2
- library functions 13-3
- library subroutines 13-2
- Library Subroutines and Functions 13-1
- library, date
 - functions 13-7
- library, EASEL OS/2 EE
 - functions 13-4
- line
 - colored 7-12
 - copying contents of 8-43
 - drawing 7-11
 - in background at statement 8-47
 - in foreground at statement 8-47
 - with textual function 8-43
- line graph 16-41
- line height of
- line size of function 4-43, 8-47
 - textual region 8-47
- list box definition
 - model 9-12
- list boxes 9-11
- literal 4-12
 - boolean 4-15
 - floating point 4-12
 - integer 4-12
 - string 4-13
- local applications
 - writing 25-1
- local entity 11-14
- loop
 - block (Program) 11-26
 - exit from 11-22, 11-24
 - nested 11-25
- loop statement 11-3, 11-17, 11-21
- lx 23-4

M

- m 23-4
- m command line option 11-15
- machine requirements 1-2, B-2
- make block statement 8-5, 8-55

- make block/segment COLOR statement
 - background keyword 8-56
 - column keyword 8-56
 - foreground keyword 8-56
 - line keyword 8-56
 - thru keyword 8-56
- make border statement 6-30
- make color statement 6-29
- make cursor invisible 8-30
- make cursor statement 8-30
- make cursor visible 8-30
- make invisible statement 5-8, 6-28
- make item parameter
 - model of 10-17
- make parameter statement 6-31
- make point disposition statement
 - drop keyword 15-7
 - mouse keyword 15-7
- make segment statement 8-5, 8-55
- make statement
 - block keyword 8-56
 - disposition keyword 15-7
 - point keyword 15-7
 - segment keyword 8-56
- make string disposition 15-9
- make string disposition statement 14-16
- make visible statement 5-8, 6-28
- manipulating 10-19
- mapped conversations 19-2
- margin statement 7-9
- marker function 4-24, 4-45
- markpar 14-21
- master
 - ancestor 5-12, 5-21
 - screen size 5-18
- master keyword 3-19
- match
 - multiple 14-13
- match clause 6-21, 15-9
- math
 - subroutines 13-11
- math library
 - ABS function 13-14
 - ARCCOS function 13-14
- math library (*continued*)
 - ARCSIN function 13-15
 - ARCTAN function 13-15
 - ARCTAN2 function 13-15
 - CEIL function 13-16
 - COS function 13-16
 - E function 13-17
 - EXP function 13-17
 - FLOOR function 13-17
 - function 13-11
 - LN function 13-18
 - LOG function 13-18
 - MAX subroutine 13-21
 - MAXINT subroutine 13-21
 - MEAN subroutine 13-21
 - MEANINT subroutine 13-21
 - MEDIAN subroutine 13-22
 - MEDIANINT subroutine 13-22
 - MIN subroutine 13-23
 - MININT subroutine 13-23
 - MOD function 13-18
 - PI function 13-19
 - POWER function 13-19
 - SIN function 13-20
 - SQRT function 13-20
 - STDDEV subroutine 13-23
 - STDDEVINT subroutine 13-23
 - SUM subroutine 13-24
 - SUMINT subroutine 13-24
 - TAN function 13-20
 - VARIANCE subroutine 13-24
 - VARIANCEINT subroutine 13-25
- math library functions 13-11
- math subroutines 13-11
 - error messages for 13-13
- mathlib.inc file A-16
- maximize attribute 3-16, 10-5
- maximize button 3-17
- mcaps keyword 5-46
- medium keyword 5-46
- medvt10 keyword 5-46
- medvt12 keyword 5-46
- medvt14 keyword 5-46
- medvt16 keyword 5-46
- medvt20 keyword 5-46

- medvt24 keyword 5-46
- member function 4-45
- members of function 4-45, 6-41
- memory
 - allocating local 11-34
 - changing allocation in EASEL.BAT 11-34
 - global 11-33
 - global and local 11-14
 - global, allocating 11-34
 - global, restrictions in use 11-33
 - global, uses of 11-33
 - handling within program 11-34
 - runtime configurations 11-33
- message boxes 13-49
- message.inc file A-18
- minimize attribute 3-16, 10-4
- minimize button keyword 3-17
- miscellaneous action routines 20-16
 - 3270 20-16
 - 5250 21-16
- mixing regions 5-39
- mnemonics
 - in action bars 10-15
- Mnemonics with dialog controls 9-7
- mod operator 4-49
- mode
 - textual line statement 9-14
- model
 - check box definition 9-20
 - checked built-in function 9-20
 - checked in group 9-20
 - group box definition 9-23
 - line statement 9-14
 - push button definition 9-17
 - radio button definition 9-19
 - selected line from function 9-14
 - selected line function 9-14
 - static text definition 9-22
- module statement 8-60
 - BGraph keyword 4-57
 - CText keyword 4-57
 - M3270 keyword 4-57
 - M5250 keyword 4-57
- module statement (*continued*)
 - 3270 20-3
 - 5250 21-3
- modules 4-57
- monitor 5-16
- mouse 15-2
 - cursor 15-2
 - pointer appearance 15-2
- mouse pointer appearance
- move arc statement 7-19, 7-21
- move statement 7-9, 7-10, 7-21
- move to statement 6-26
- moving
 - graphics cursor 7-10
 - text cursor 7-10
- Multilingual code page 4-4
- multiple-line entry fields 9-9
- M3270 module 20-2
- M5250 module 21-2
- M5250 Support
 - watch facility 21-33
- M52_AutoWait variable 21-52
- M52_CheckIndicators 21-56
- M52_Connect 21-19
- M52_DefineWatch 21-39
- M52_Disconnect 21-20
- M52_EnterString 21-54
- M52_FindLastNonBlankLine 21-29
- M52_GetCursorPosition 21-25
- M52_GetFieldAttributes 21-24
- M52_GetFieldLength 21-23
- M52_GetFieldNumber 21-26
- M52_GetFieldPosition 21-27
- M52_GetSessionInfo 21-57
- M52_GiveUpTime 21-45
- M52_InformWhenIloff 21-34
- M52_Init 21-17
- M52_PressCMDkey 21-52
- M52_PressENTER 21-51
- M52_PressKey 21-49
- M52_PrintScreen 21-59
- M52_ReadColoredScreen 21-31
- M52_ReadField 21-22
- M52_ReadLine 21-28
- M52_ReadScreen 21-30

- M52_ScanScreen 21-21
- M52_Stop 21-18
- M52_TypeString 21-53
- M52_WaitForIloff 21-36
- M52_Watch 21-45
- M52_WatchAndWait 21-47
- M52_WatchCommandString 21-41
- M52_WatchForChar 21-43
- M52_WatchForCursorAt 21-40
- M52_WatchForCursorNotAt 21-41
- M52_WatchForIloff 21-42
- M52_WatchForNotChar 21-44
- M52_WriteField 21-55

N

- names command 17-23
- National Language Support
 - code page 4-4
 - date and time 4-7
 - keyboards 4-5
 - languages 4-4
- nb 23-10
- nested
 - pattern 7-30, 7-35
- nested blocks 6-10, 6-12, 6-15
- nested include files 11-8
- new-line character 8-16, 8-34
 - overwriting 8-22
- no scale attribute 10-7
- no scale keyword 3-17
- non-string data, sending/receiving
 - in APPC 19-15
- nopar 14-21
- Nordic code page 4-4
- normal 14-16
- normal keyword 15-7
- notation conventions v
- null command 17-10
- number 14-11
- number in match clause 14-9

O

- object
 - adding to class 6-34, 6-37, 6-38
 - ancestry 5-12
 - class membership 6-38
 - color 5-8
 - components of 5-3
 - coordinate system 5-18, 7-10
 - creating 6-24
 - creating identifiers 7-3
 - definition 3-7
 - deleting 6-28
 - deleting from class 6-38
 - name 5-3
 - origin 5-18
 - parameter 5-15
 - position 5-3
 - priority 5-14
 - selectability 5-5
 - siblings 5-12
 - type 5-3
 - types of 5-2
 - visibility 5-8
- object contents
 - adding 6-25
 - deleting 6-26
 - replacing 6-27
- object function 4-42, 6-7, 6-8, 8-43
 - with a class 6-39
 - with ancestry function 6-39
 - with parameter function 6-40
- objects 3-8
 - graphical 7-2
- oddpair 14-21
- open statement 15-6, 15-7
- open subroutine 13-31
 - error messages 13-34
- operands 4-47
- operating environment 1-2, B-1
- operators 4-47
 - arithmetic, precedence in 4-49
 - arithmetic, table of 4-48
 - boolean, precedence in 4-51
 - boolean, table of 4-51
 - relational, precedence in 4-51

- operators (*continued*)
 - relational, table of 4-50
- OPTION_STRING parameters 22-3
- OS/2 Command File
 - COMPILE.CMD 23-3
- output command 17-13
- output devices 22-2
- output subroutines 13-41
 - error messages for 13-43
- overlapping bar graph 16-10, 16-45
- Overview 13-2
- overview of APPC support 19-2
- overview of audio support 18-2
- overwrite statement
 - column keyword 8-20
 - line keyword 8-20
 - segment keyword 8-20
 - text cursor 8-21
- overwriting
 - string 8-21

P

- parameter attribute 5-15
- parameter function 4-22, 4-42, 5-15, 6-7, 6-31, 8-42, 11-17
 - with object function 6-40
- parameter of function 4-43, 5-15, 6-40, 7-61, 8-42
- parent
 - valid types 5-13
- parm adjustment subroutines
 - error messages for 13-46
- pattern
 - box statement 7-38
 - color 7-35
 - coordinate system 7-34
 - definition 3-7
 - draw statement 7-34
 - font in a 7-36
 - nested 7-30, 7-39
 - reference point of 7-33
 - rotate specification 7-37
 - rotating a 7-37
 - scale factor 7-36
 - scale specification 7-36
- pattern (*continued*)
 - text 7-39
- pattern is statement 7-30
- pattern statement 7-32
- patterns
 - nested 7-35
- pie charts
 - Business Graphics 7-18
- pie graph 16-11, 16-43
- pixels 5-43
- plot action statement 22-3
- plotting
 - error messages 22-4
- plotting
 - EASEL OS/2 EE regions 22-2
 - hardcopy support 22-2
 - OPTION_STRING
 - parameters 22-3
 - output devices 22-2
 - plot action statement 22-3
- PM fonts 5-47
- pmptr.inc file A-20
- pointer attribute 10-7
- POINTER command 1-4
- pointer keyword 3-17
- pointing device
 - as global entity 11-14
 - default 15-6
 - mouse 15-2
 - touchscreen 15-2
- pointing devices 3-20
 - overview 15-2
 - set pointer to statement 15-2
- polygon
 - color of 7-15
 - colored 7-15
 - statement 7-14
- polygon statement 7-9
 - definition 7-14
- portability B-5
- Portuguese code page 4-4
- position
 - inquiry 7-54
- precision statement 4-13
- PressENTER 20-46

- PressKey 20-45
- PressPFKey 20-47
- PRI file 4-58
- primary attribute 3-19
- primary keyword 3-8, 3-17
- printing
 - ASCII files 22-5
 - EASEL OS/2 EE regions 22-2
 - error messages 22-7
 - example 22-6
 - hardcopy support 22-2
 - optional parameters 22-5
 - output devices 22-2
 - utilities 22-5
- Print3270Screen 20-59
- priority attribute 5-14, 7-73
- priority of function 4-43, 6-31, 7-62, 8-42
- profiles, APPC
 - description 19-12
 - overview 19-2
 - setting/querying 19-12
 - subroutines 19-13
- program 3-6
 - compile 3-15
 - components, order 3-6
 - constructing 3-8
 - control 11-8
 - execute 3-15
 - sample 3-9
 - textual regions 8-49
- program construction 3-6
- program flow 11-2
- program flow statements 11-4, 11-8, 11-17
- program requirements 1-3, B-4
- prompt command 17-8
- PUB file 4-57
- pulldown action
 - model of 10-11
- pulldowns
 - defining 10-11
 - defining choices 10-12
- push buttons 9-17

R

- radio buttons 9-18
 - check statement 9-19
 - uncheck statement 9-19
- read statement 7-70, 8-20
 - colored keyword 8-33, 9-13
 - file keyword 8-33, 9-13
 - ioerror function 4-44, 8-33
 - model 9-13
 - text cursor 8-33
- ReadColoredScreen 20-31
- ReadField 20-22
- ReadLine 20-28
- ReadScreen 20-30
- redrawing
 - specified object 5-52
- refresh statement 5-52
- region
 - mixing types 5-39
 - multiple 5-31
 - position attribute 5-23
 - size attribute 5-23
- region actions 10-8
- Region Attributes 10-2
 - model of 10-3
- Region Attributes and Action Bars 10-1
- region keyword 3-8
- relative position
 - in coordinate system 5-35
- remove statement
 - from keyword 9-13
 - line 8-28
 - model 9-13
 - text cursor 8-28
- removing
 - segment 8-28
- required procedures
 - 3270 20-3
 - 5250 21-3
- reserved constants
 - 3270 20-74
 - 5250 21-70
- reserved constants, APPC 19-98

- reserved constants, audio 18-33
- reserved variables
 - 3270 20-71
 - 5250 21-67
- resize statement 4-40
- resolution
 - monitor 5-17
- resolution-dependent font 5-45
- responding to
 - action bar items 6-6
 - application input 6-8
 - class 6-35, 6-36
 - entering a block 6-3
 - keyboard input 6-8
 - mouse double click 6-4
 - objects 6-4
 - program startup 6-3
 - stimuli 6-17, 6-19
 - stimuli types 6-3
 - system menu close 6-4
 - window activation 6-4
- response
 - definition 3-7
- response inquiry
 - built-in functions 6-7
- response statement 3-9
- response to
 - char 6-2, 6-8
 - interval 6-3
 - line 6-2, 6-8
 - low memory 6-3, 6-17, 6-19
 - object 6-2
 - object class 6-35
 - overview 6-2
 - start 6-2, 6-19
 - stimulus 6-2
 - termination 6-3, 6-17, 6-19
 - timeout 6-3
- response to char 14-13
- response to char statement 14-2, 14-7, 14-9, 15-9
- response to interval statement 11-29
- response to item
 - model 10-27

- response to item statement 6-6
- response to line 14-8, 14-13
- response to line statement 14-2, 14-7, 15-9
- response to low memory statement 11-34, 11-38
- response to object statement 6-4, 8-26
- response to start 11-11
- response to start statement 6-3
- response to stimulus
 - definition 14-29
- response to termination statement 6-16
- response to timeout statement 11-30
- responses 3-8
- resumable block
 - definition 6-11
 - description 6-17
- right of function 4-43, 7-55, 8-41
- rotating
 - pattern 7-37
 - specifications 7-37
 - text 7-39

S

- sample keyword 7-67
- sample programs
 - audio support 18-31
 - automatic interface 19-93
 - controlled interface 19-95
- save as program statement 11-2
- save program as statement 4-18, 11-10, 11-12
 - to save program in progress 11-15
 - with global and local values 11-15
- scalable font 5-50
- scaling
 - in image regions 7-67
 - pattern 7-36
- ScanScreen 20-21

- scatter graph 16-44
- screen
 - ancestry of 5-21
 - coordinate system 5-16
 - origin 5-17
 - resolution 5-17
- Screen Size 3-16, 7-56
- screen size declaration 3-6
- screen size Environment Declaration 3-16
- screen size statement 5-16, 5-21
- scroll attribute 10-6
- scroll bar attribute 3-16
- segment
 - copying contents of 8-43
 - in color header 8-60
 - in colored textual regions 8-55
 - text 8-17
 - textual region 8-10
 - with textual function 8-44
- selectability of function 4-43, 6-28, 7-62, 8-42
- selected line from function 4-43
- selected line function 4-45
- send statement 4-45, 14-2, 14-7, 24-4
 - errorlevel function 4-45
- sending/receiving non-string data in APPC 19-15
- sending/receiving string data in APPC 19-14
- Sense regions 7-72
- senseport 5-5
- separator definition
 - model of 10-13
- separators
 - defining 10-13
- serial number 1-3
- set command (OS/2) 3-14
- set low memory threshold
 - statement 11-34, 11-36
- set pointer to statement 15-2
- SetEmulatorWindow 20-50
- shape
 - border of 7-29
 - color of 7-29
- shape (*continued*)
 - solid or hollow 7-28
- shape statement 7-9, 7-15, 7-19
- shapes
 - graphics cursor 7-23
- shift keyword 10-15
- single-line entry fields 9-8
- size
 - considerations 3-21
 - inquiry 7-54
 - viewports 8-4
 - windows 8-4
- size attribute 7-4
- SKIP_CLAUSE 4-23
- sound 15-14
- spacepar 14-21
- SQL Support
 - access plans 17-2
 - case of characters 17-32
 - command line options 17-30
 - command types 17-2
 - configuration commands 17-7
 - direct commands 17-3
 - dynamic commands 17-26
 - dynamic commands
 - requirement 17-32
 - error handling 17-27
 - error message 17-32
 - graphic data types 17-33
 - increasing memory 17-32
 - input 17-2
 - input command 17-31
 - inquiry commands 17-21
 - isolation level 17-32
 - LONG VARCHAR 17-31
 - notes 17-31
 - output 17-2
 - output, using 17-31
 - records, number of 17-32
 - requirements 17-2
 - standard error 17-32
 - statement size 17-32
 - strings, customizing 17-31
 - white space 17-32
- SQL Support commands
 - binddb 17-6

SQL Support commands (*continued*)

- bindfile 17-12
- display 17-22
- error 17-9
- exit 17-20
- fs 17-14
- info 17-11
- maxwidth 17-17
- maxwidth command 17-17
- names 17-23
- null 17-10
- output 17-13
- prompt 17-8
- rs 17-15
- rs command 17-15
- set 17-16
- set command 17-16
- startdb 17-4
- startq 17-18
- stopdb 17-5
- types 17-24
- verbose 17-19
- widths 17-25
- squeeze memory statement 11-35, 11-37
- stacked bar graph 16-46
- stacked line graph 16-42
- standalone program 25-2
- standard bar graph 16-44
- standard line graph 16-41
- start local 14-4
- start remote 14-5
- start statement 14-4
 - errorlevel function 4-45
- start statement start action 14-2
- startdb command 17-4
- starting
 - 3270 20-14
 - 5250 21-14, 21-17
- startq command 17-18
- statement 8-25
- statement(s)
 - variable 3-7
- static text 9-21
- stepped line graph 16-14, 16-42

stimulus

- declaration, audio 18-4
- declaration, in APPC 19-5
- events in APPC, responding to 19-4, 19-8
- events, audio, responding to 18-4
- stimulus declaration 14-29
- Stimulus libraries 14-29
- stimulus response 14-29
- stop all statement 14-6
- stop statement 14-2, 14-6
- stopbits 14-22
- stopdb command 17-5
- stopping
 - 3270 20-14
 - 5250 21-14, 21-17
- Stop3270 20-18
- string
 - finding 8-34
- string data, sending/receiving in APPC 19-14
- strings 25-5
- subroutine declaration 11-2
- subroutine is definition 11-2
- subroutine is statement 11-6
- subroutines 11-4
 - open and close 13-31
 - output 13-41
 - writing 25-3
- supported 7-63
 - bitmap file formats 7-63
- surface line graph 16-43
- suspend processing 11-31
- synchronization
 - 3270 20-14
 - 5250 21-15
- syntax
 - frame oriented 3-16
- system menu accelerators
 - use in fullscreen applications 3-21
- system menu attribute 3-16, 10-5
- system menu keyword 3-17
- system monospaced font 5-47

system proportional font 5-47

T

tab character 8-16, 8-17, 8-34

TAKE_CLAUSE 4-23

-terp 23-10, 24-3, 24-5

text

- aligning 7-45

- centering 7-45

- changing disposition of 14-16

- color of 7-46

- coloring 8-55, 8-58

- converting to uppercase 5-46

- deemphasizing 8-30

- displaying in graphical

 - objects 7-44

- displaying outline in a graphical region 7-45

- emphasizing 8-29, 8-30, 8-55

- font of 7-46

- overwriting 8-20

- removing 8-28

- rotating 7-39

text align statement 7-45

text center statement 7-45

text cursor 5-35

- in textual region 5-37

- visibility of 8-30

text of function 4-43, 8-42

text of item function 4-44

text statement 7-9, 7-44

- font statement 7-46

textual built-in function 8-42

textual function 4-43, 8-43

- column keyword 8-43

- from keyword 8-43

- line keyword 8-43

- segment keyword 8-43

- thru keyword 8-43

- with extract from

 - statement 8-46

- with insert statement 8-44, 8-45

textual line from function 4-43

textual region 8-5

- action statements 8-27

textual region (*continued*)

- add to statement 8-28

- addressing a character in 8-8

- ancestry 8-2, 8-38

- appending to a file 8-31

- attributes 8-4

- colored 8-5

- columns and lines in 8-46

- coordinate system 5-20, 8-8

- coordinate system of 8-2

- defining 8-1, 8-2

- definition 8-4

- drawing statements 8-2, 8-7

- finding text in 8-34

- fonts 8-3, 8-6, 8-33

- foreground color 5-11

- move statement 8-9

- overwriting 8-23

- overwriting a file with 8-32

- program, sample 8-49

- reading a file, colored 8-33

- reading to 8-32

- response definitions 8-26

- segment, colored 8-10

- senseport 8-26

- size 8-4

- text cursor 5-37, 8-9

- text cursor in 8-2

- viewport 8-4, 8-36

- window 8-4, 8-36

- window of 8-2

- window size 8-4

- window, changing size 8-38

- writing to a color-attributed

 - file 8-32

- writing to a file 8-30

textual region definition

- color keyword 8-54

- default keyword 8-54

- foreground keyword 8-54

- position keyword 8-54

textual regions

- Block (Text), colored 8-12

- built-in Functions 8-41

textual region, changing position

- viewport 8-37

- textual region, changing position
 (continued)
 - window 8-39
- textual region, changing size
 viewport 8-37
- textual region, reducing size
 viewport 8-37
- three dimension b 16-10
- 3-D stacked bar graph
- three dimensional bar graph 16-10
- three-dimension 16-45
- 3-D Bar Graph
- three-dimension o 16-11
- 3-D overlapping bar graph 16-46
- time function 4-45
- timing
 Program 11-29
- title bar attribute 3-16, 10-4
- title bar keyword 3-17
- tms rmn font 5-48
- to keyword 5-35
- top of function 4-43, 7-55, 8-41
- touchscreen 15-2
- touchscreen keyword 15-7
- tracing, enabling APPC 19-101
- turn trace action statement 24-3
- turn trace off statement 24-3, 24-5
- turn trace on statement 24-3, 24-5
- turn trace statement 24-5
- type conversions 4-53
- types command 17-24
- TypeString 20-48

U

- uncheck keyword 10-22
- update screen
 wait 0 statement 11-32
- uppercase
 conversion to 5-46
 font 5-46
- utilities
 printing 22-5
- utility subroutines, APPC 19-22
- U.S. code page 4-4

V

- value
 boolean 11-22
 compile-time 4-11
 constants 4-16
 literals 4-12
 string, copying to a 8-43
 type conversion 4-53
 types 4-10
 variables 4-17
- value in match clause 14-10
- variable
 arrays 11-14
 assigning value(s) to 4-17, 4-20
 global 11-14
 global and local 4-18
 initializing 4-17
 string, finding length of 4-19
- variable statement 4-17
- verbose command 17-19
- vertical scroll bar keyword 3-17
- VGA
 fullscreen colors 5-9
- viewport 5-23
 ancestry 5-30
 change size statement 7-48
 changing size of 7-48
 graphical region 7-48
 in graphical region 7-47
 region 5-28
- viewport, changing position of
 graphical region 7-50
- visibility
 of text cursor 8-30
- visibility attribute 7-73
- visibility of function 4-43, 6-28, 7-62, 8-42
- visible keyword 5-8

W

- wait for statement 11-31
- wait statement 5-52, 11-31
- Watch 20-41

- WatchAndWait 20-43
- WatchForChar 20-39
- WatchForCursorAt 20-36
- WatchForCursorNotAt 20-37
- WatchForNotChar 20-40
- WatchForNoX 20-38
- wedge
 - color of 7-19
 - colored 7-19
 - degrees 7-18
 - limitations 7-19
 - solid 7-19
 - solid or hollow 7-19
 - statement 7-18
- wedge statement 7-9, 7-18
 - definition 7-18
- welcome.eal. 3-11
- while clause
 - in loop statement 11-21
- widths command 17-25
- window 5-22
 - ancestry 5-30
 - changing position of 7-52
 - changing size of 7-51
 - graphical region 7-51, 7-52
 - in graphical region 7-47
 - region 5-24
- window xposition of function 4-43, 7-58, 8-43
- window xsize of function 4-43, 7-57, 8-42
- window yposition of function 4-43, 7-58, 8-43
- window ysize of function 4-43, 7-58, 8-43
- windowed application
 - scaling 3-20
- windowed applications
 - architecture 3-21
- windowed vs fullscreen
 - applications 3-19
- with textual function
 - from keyword 9-14
 - thru keyword 9-14
- word 14-11

- word in match clause 14-9
- word wrap
 - in multiline entry field
 - definition 9-10
- work area cache 7-66
- write statement 7-70
 - append keyword 8-30, 8-31
 - colored file keywords 8-30
 - destructive 8-32
 - destructive keyword 8-30
 - file keyword 8-30
 - ioerror function 4-44
 - ioerror function with 8-31
- WriteField 20-56
- writing
 - DLLs 25-3
 - functions 25-3
 - guidelines 25-2
 - keyboard 25-2
 - libraries 25-1, 25-3
 - local applications 25-1
 - screen I/O 25-2
 - standalone program 25-2
 - subroutines 25-3
 - well-behave applications 25-2

X

- xcoord function 4-42, 6-7, 6-9, 8-26, 8-41
- xcursor of function 4-43, 7-56, 8-35, 8-41
- xmiddle of function 4-43, 7-55, 8-41
- XNumTicks 16-23
- xposition of function 4-43, 7-57, 8-41
- xsize of function 4-43, 7-57, 8-41
- XTicks 16-23

Y

- ycoord function 4-42, 6-7, 6-9, 8-26, 8-41
- ycursor of function 4-43, 7-56, 8-35, 8-41

- ymiddle of function 4-43, 7-56, 8-41
- YNumTicks 16-23
- yposition 7-57
- yposition of function 4-43, 8-41
- ysize of function 4-43, 7-57, 8-42
- YTicks 16-23

Z

- zooming
 - in 5-25
 - out 5-25

Numerics

- 3270 action routines
 - Connect 20-19
 - DefineWatch 20-35
 - Disconnect 20-20
 - EmulateAndWatch 20-53
 - Emulate3270 20-52
 - EnterString 20-49
 - FindLastNonBlankLine 20-29
 - GetCursorPosition 20-25
 - GetFieldAttributes 20-24
 - GetFieldLength 20-23
 - GetFieldNumber 20-26
 - GetFieldPosition 20-27
 - GetSessionInfo 20-62
 - GetSessionStatus 20-60
 - GetXstatus 20-57
 - Init3270 20-17
 - PressENTER 20-46
 - PressKey 20-45
 - PressPFKey 20-47
 - Print3270Screen 20-59
 - ReadColoredScreen 20-31
 - ReadField 20-22
 - ReadLine 20-28
 - ReadScreen 20-30
 - ScanScreen 20-21
 - SetEmulatorWindow 20-50
 - Stop3270 20-18
 - TypeString 20-48
 - Watch 20-41
 - WatchAndWait 20-43

- 3270 action routines (*continued*)

- WatchForChar 20-39
- WatchForCursorAt 20-36
- WatchForCursorNotAt 20-37
- WatchForNotChar 20-40
- WatchForNoX 20-38
- WriteField 20-56

- 3270 Support 20-19

- A3270.EXE 20-2
- buffer, examining 20-21
- communicating with host 20-45
- components 20-2
- Connecting 20-19
- controlling emulator
 - window 20-50
- Disconnecting 20-19
- error handling 20-64
- error messages 20-66
- error status variables 20-64
- fields utility 20-2, 20-8
- miscellaneous action
 - routines 20-56
- module statement 20-3
- M3270 action routines 20-13
- M3270 module 20-2
- program sample 20-4
- program sample, screens 20-75
- referencing M3270 20-3
- required procedures 20-3
- reserved constants 20-74
- reserved variables 20-71
- screen size 20-6
- screen, field-orientated 20-6
- screen, line-orientated 20-6
- sessions 20-5
- starting 20-17
- stopping 20-17
- synchronizing 20-33

- 5250 action routines

- M52_CheckIndicators 21-56
- M52_Connect 21-19
- M52_DefineWatch 21-39
- M52_Disconnect 21-20
- M52_EnterString 21-54
- M52_FindLastNonBlankLine 21-29
- M52_GetCursorPosition 21-25

5250 action routines (*continued*)

- M52_GetFieldAttributes 21-24
- M52_GetFieldLength 21-23
- M52_GetFieldNumber 21-26
- M52_GetFieldPosition 21-27
- M52_GetSessionInfo 21-57
- M52_GiveUpTime 21-45
- M52_InformWhenIloff 21-34
- M52_Init 21-17
- M52_PressCMDkey 21-52
- M52_PressENTER 21-51
- M52_PressKey 21-49
- M52_PrintScreen 21-59
- M52_ReadColoredScreen 21-31
- M52_ReadField 21-22
- M52_ReadLine 21-28
- M52_ReadScreen 21-30
- M52_ScanScreen 21-21
- M52_Stop 21-18
- M52_TypeString 21-53
- M52_WaitForIloff 21-36
- M52_Watch 21-45
- M52_WatchAndWait 21-47
- M52_WatchCommandString 21-41
- M52_WatchForChar 21-43
- M52_WatchForCursorAt 21-40
- M52_WatchForCursorNotAt 21-41
- M52_WatchForIIOff 21-42
- M52_WatchForNotChar 21-44
- M52_WriteField 21-55

5250 Support

- action routines 21-13
- action routines, summary 21-14
- A5250.EXE 21-2
- communicating with host 21-49
- components 21-2
- connecting 21-19
- disconnecting 21-19
- error handling 21-60
- error messages 21-62
- examining the buffer 21-21
- field-oriented 21-6
- fields utility 21-3, 21-9
- Input Inhibited indicator 21-33
- line-oriented screens 21-6
- module statement 21-3

5250 Support (*continued*)

- M5250 module 21-2
- program sample 21-4
- referencing the module 21-3
- required procedures 21-3
- reserved constants 21-70
- reserved variables 21-67
- retrieving information 21-9
- session names 21-5
- session states 21-5
- starting 21-17
- stopping 21-17
- synchronizing via 3270 device emulation 21-37
- synchronizing with AS/400 21-34
- synchronizing with host 21-33
- using 5250 screens 21-71
- watch facility 21-37

Special Characters

- .CMD file 24-5
- .EAL file 3-12
- .EBI file 24-2

Developer's Guide

SC38-7034-01

Reader's Comment Form

Please use this form only to identify publication errors or to request changes in publications. Direct any requests for additional publications, or technical information about IBM systems, to your IBM representative or nearest IBM branch office.

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Comments:

If you wish a reply, give your name, company, mailing address, and date.

Name _____
Company _____
Address _____
City _____ State _____ Zip Code _____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)



**NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES**

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40

ARMONK, NEW YORK 10504

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
10401 Fernwood Road
Bethesda, MD 20817



ATTENTION: Software Development, Dept. 877

.....
Fold Here

Tape

Please Do Not Staple

Tape

Continued from inside front cover

SUCH WARRANTIES ARE IN LIEU OF ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

LIMITATION OF REMEDIES

IBM's entire liability and your exclusive remedy shall be as follows:

1. IBM will provide the warranty described in IBM's Statement of Limited Warranty. If IBM does not replace defective media or, if applicable, make the Program operate as warranted or replace the Program with a functionally equivalent Program, all as warranted you may terminate your license and your money will be refunded upon the return of all of your copies of the Program.
2. For any claim arising out of IBM's limited warranty, or for any other claim whatsoever related to the subject matter of this Agreement, IBM's liability for actual damages, regardless of the form of action, shall be limited to the greater of \$5,000 or the money paid to IBM, its Authorized Dealer or its approved supplier for the license for the Program that caused the damages or that is the subject matter of, or is directly related to, the cause of action. This limitation will not apply to claims for personal injury or damages to real or tangible personal property caused by IBM's negligence.
3. In no event will IBM be liable for any lost profits, lost savings, or any incidental damages or other conse-

quential damages, even if IBM, its Authorized Dealer or its approved supplier has been advised of the possibility of such damages, or for any claim by you based on a third party claim.

Some states do not allow the limitation or exclusion of incidental or consequential damages so the above limitation or exclusion may not apply to you.

GENERAL

You may terminate your license at any time by destroying all your copies of the Program or as otherwise described in this Agreement.

IBM may terminate your license if you fail to comply with the terms and conditions of this Agreement. Upon such termination you agree to destroy all your copies of the Program.

Any attempt to sublicense, rent, lease or assign, or, except as expressly provided herein, to transfer any copy of the Program is void.

You agree that you are responsible for payment of any taxes, including personal property taxes, resulting from this Agreement.

No action, regardless of form, arising out of this Agreement may be brought by either party more than two years after the cause of action has arisen except for breach of the provisions in the Section entitled "License" in which event four years shall apply.

This Agreement will be construed under the Uniform Commercial Code of the State of New York.

Z125-3301-02 4/87
Printed in U.S.A.



SC38-7034-01



Printed in U.S.A.